Specification-Based Testing with Buchi Automata: Transition Coverage Criteria and Property Refinement

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

***Abstract*—Büchi automaton is instrumental in linear-temporal logic model checking. It has been used in formalizing linear temporal requirements as well as in designing model checking algorithms. In this work we extend Büchi automaton to the domain of specification-based testing. We developed test criteria and techniques essential for testing a system with a formal requirement in Büchi automata. At the core of our approach are two Büchi-automaton-based test criteria that select test cases based on their relevancy to a requirement in Büchi automaton. The relevancy is based on the notion of transition coverage on Büchi automaton. We define "weak" and "strong" variants of transition coverage criteria that reflect the non-deterministic nature of a Büchi automaton. The new transition coverage criteria out-perform state coverage criteria in [1], both on theory and in our experiments. Our experiment demonstrates the effectiveness of the proposed transition coverage criteria by measuring cross-coverage of these transition coverage criteria versus other existing test criteria. To improve test efficiency, we provide model-checking-assisted algorithms that fully automate test vector generations for the transition coverage criteria. In addition, we propose property refinement using the feedback from the test generation algorithm. The benefits of our approach are two-fold: (1) it enables the effective and efficient testing with formal requirements in Büchi automata; and, (2) our approach is capable of not only finding bugs in a system, but also identifying deficiency in its requirement via property refinement.**

I. Introduction

Testing and formal verification are two commonly used verification and validation (V&V) techniques. They provide benefits that are often complementary to each other. Testing checks the behaviors of a system under controlled input stimuli, also known as test cases. As a traditional V&V technique, testing often plays a central role in a V&V process. Testing is also an important component of many software quality standards. For example, DO-178, the software quality standard for safety-critical avionic software, has set the testing requirements for various structural criteria, including the MC/DC criterion [2]. Heimdahl *et. al* [3] estimated that V&V activities can consume 50%-70% of resource in developing high-dependable software, and much of the cost are incurred during testing. One drawback of testing, as Dijkstra once famously noted, is "testing shows the presence, not the absence of bugs". A constant theme in software engineering research is how to make testing more effective and efficient.

Formal verification, on the other hand, is the technique to build a mathematically rigid proof for the correctness of a system with respect to its specifications. Formal verification, especially model checking (cf. [4]), has enjoyed great successes in industry practices. With increasing acceptance of formal verification, formal specification of high quality has become common, and an array of efficient formal verification tools have been developed.

Given their pros and cons, testing and formal verification would likely co-exist as two most important V&V techniques in the foreseeable future. A theme of our research is *how to harness the synergy of both techniques to build more efficient and effective V&V processes*. In this research we address this research challenge in context of specification-based testing with Büchi automaton. Büchi automaton is a form of ω-automata. It accepts ω-language, an extension of regular languages with infinite words. Büchi automaton has been instrumental in developing linear temporal model checking. It has been used to specify linear temporal properties of a system, and to develop efficient linear temporal model checking algorithms [5], [6]. Encoding requirement in a rigorous formalism such as Büchi automaton helps remove ambiguity in system specification and facilitates formal verification. With demands for high-dependable systems and a surge of popularity of formal verification in recent years, there are an increasing amount of system requirements specified in a formal language such as Büchi automaton and its related temporal logic LTL (Linear Temporal Logics). A research question is *how to efficiently and effectively test a system with its requirements specified in Büchi automata.*

In this work we developed a set of techniques that enable the efficient and effective testing of a system with a formal requirement in Büchi automaton. To improve the effectiveness of testing, we center testing around the formal requirement. Specifically, we relate testing to a Büchi automaton by introducing transition coverage metrics and criteria. A transition coverage metric measures how well a test suite covers the transitions of a Büchi automaton. To reflect the undeterministic nature of a Büchi automaton, we define "weak" and "strong" variants of transition coverage metrics.

To improve the efficiency of testing, we developed model-checking-assisted test generation algorithms for the proposed transition coverage criteria. We implemented these algorithms on an off-the-shelf linear temporal model checker SPIN [7]. At the core of these algorithms are graph-transformation techniques that transform a Büchi automaton to a set of "trap"

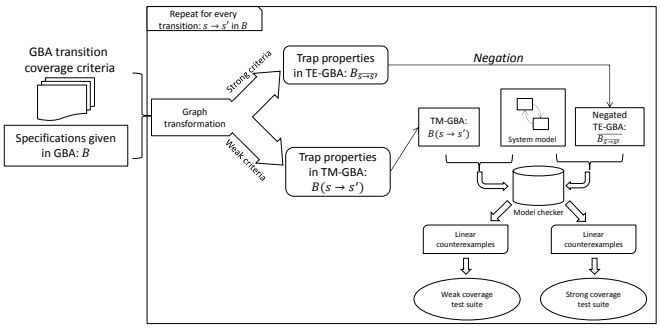

Fig. 1. The workflow of model-checking-assisted test case generation under transition coverage criteria for Büchi automaton.

automata. Each "trap" automaton specifies a test objective that covers a particular transition of the automaton. Figure 1 shows the workflow of model-checking-assisted test case generation under transition coverage criteria for Büchi automaton.

Our proposed approach also extends the benefits of testing from debugging to improving requirements. We developed a requirement refinement technique using the information from test case generation. This closes an important feedback loop between testing and requirement: normally requirements drive testing activities; using our approach, testing also helps refine a requirement.

We conduct a systematical computation study to compare the effectiveness of the proposed transition coverage criteria versus those of other existing criteria, including traditional structural and logical testing criteria, as well as the state coverage criteria for Büchi automaton. The effectiveness is measured by the cross-coverage between the transition coverage metrics and other existing testing metrics. The result of our computational study clearly indicates the effectiveness of our transition coverage criteria.

The rest of the paper is organized as follows: Section II defines the notations that will be used in the rest of the paper; Section III introduces two variants of transition coverage metrics and criteria for Büchi automata; Section IV describes the model-checking-assisted test case generation algorithms for the transition coverage 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.

A. Related Works

As part of our approach, we use model checkers to automate test generation for transition coverage criteria. Model-checking-assisted test case generation attracted much research interest in recent years. It uses highly efficient model checking algorithms to improve the efficiency of test case generation. In [8] Ammann and Black defined a specification-based coverage metric. Their primary motivation is to evaluate test suites against the state/transition specifications used with model checking. Our motivation is to develop coverage criteria that may be used to automate test generation for specification-based testing, and the formalism used in our study, Büchi-automaton, enables us to specify more complicate temporal properties.

A central question in model-checking-assisted test generation is how to formulate test objectives as temporal properties accepted by a model checker. It was shown that traditional structural coverage criteria such as MC/DC Coverage can be encoded in Computational Tree Logic (CTL) (cf. [9]), which can be used by a model checker such as NuSMV for generating tests. In [10] Calvagna *et al.* encoded several combinatorial testing criteria in Linear Temporal Logic (LTL), which was then used by the model checker SAL to generate tests. In [11] Hong *et al.* expressed the dataflow criteria in CTL. These previous works emphasize on applying model-checking-assisted test generation to existing testing criteria. In contrast, in [12] we considered test generation with temporal logic requirements. We proposed a coverage metric measuring how well a test covers a linear temporal logic requirement. Our coverage metric in [12] is inspired by the notion of (non-) vacuity in [13]. Intuitively the vacuity-based coverage criterion requires a test suite to check the relevancy of each subformula of a LTL property to a system. A similar strategy was described in [14], [15]: the authors proposed a Unique-First-Cause (UFC) coverage derived from the MC/DC criterion. They define the satisfying paths over LTL temporal operators, setting up a rigorous notion of requirements coverage over execution traces. For a comparison of these techniques, interested readers may refer to [9].

In [16], Fujiwara *et al.* proposed the partial W-method as the method for test case selection. It used finite state machine coverage as a criterion to determine whether the test suite is adequate or not. While our approach share the similar reasoning of their works, the core difference lies in the fact that Fujiwara *et al.* based their test case selection standard upon the model, i.e., the system being tested. What we are proposing, however, focuses on the requirements depicted in the form of Büchi automaton. We incorporate the FSM model for the system being tested as well, but we place our attention on the behaviors of the system, whose characteristics are described by the infinite-words accepting Büchi automaton.

This work can also be seen as an extension of our previous work in [12]. Whereas our vacuity-based coverage metric characterizes test cases in their relevancy to temporal specifications in LTL, a feature but also a critic of the metric is that it heavily depends on syntactical structures of LTL formulae. For example, the LTL formula $f_0 : \mathbf{G}(brake \Rightarrow \mathbf{F}\ stop) \wedge (brake \Rightarrow \mathbf{F}\ stop)$ is semantically equivalent to $f_1 : \mathbf{G}(brake \Rightarrow \mathbf{F}\ stop)$. Yet for vacuity-based coverage metric, the coverage of a test case for f_0 always subsumes its coverage for f_1. Defining coverage metrics based on Büchi automata helps alleviate this syntactical dependency. Moreover, there are several existing algorithms for minimizing Büchi automata, which can be used as a pre-process to further reduce syntactical variance of otherwise semantically equivalent Büchi automata. In addition, in this paper we will also discuss property refinement based on the proposed coverage metrics.

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 2.1 formally defines Kripke structures.

Definition 2.1 (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 i-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 2.2 (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 2.3 (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 if and only if it passes every test in ts.

B. Generalized Büchi Automata

Definition 2.4: 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. That is, it 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 appearing for infinite times on ρ. A successful run of B is a run of B such that for every $F \in \mathcal{F}$, $\inf(\rho) \cap F \neq \emptyset$.

To accept infinite words, we extend Definition 2.4 with an alphabet, which is a powerset of atomic propositions $\mathcal{A}$. In this work we extend Definition 2.4 using state labeling approach in [5] with one modification: we label the state with a set of literals, instead of with a set of sets of atomic propositions in [5]. 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 $\Lambda(L) = \{A \subseteq \mathcal{A} \mid (A \supseteq (L \cap \mathcal{A})) \wedge (A \cap (L \cap \mathcal{A}_\neg) = \emptyset)\}$, 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 2.5 for (labeled) generalized Büchi automata (GBA).

Definition 2.5: 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 involves 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 outputs an accepting trace of the product automaton, which serves as a counterexample to $K \models \phi$.

III. TRANSITION COVERAGE METRICS AND CRITERIA

Definition 3.1 (Covered Transitions): Given a generalized Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, a test t weakly covers a transition $s \rightarrow s'$ if B has a successful run ρ such that $t \vdash \rho$ and ss' is a substring of ρ. A test t strongly covers a transition $s \rightarrow s'$ if $t \models B$, and for B's every successful run ρ such that $t \vdash \rho$, ss' is a substring of ρ.

Since a generalized Büchi automaton B may have nondeterministic transitions, B may have more than one successful run induced by a test. A weakly covered transition $s \rightarrow s'$ shall appear on some successful runs induced by t, whereas a strongly covered transition $s \rightarrow s'$ has to appear on every successful run induced by t. By imposing the additional requirement that t shall also satisfy B, Definition 3.1 also requires that at least one of such successful runs exists for a transition $s \rightarrow s'$ strongly covered by t.

Definition 3.2 (Weak Trans. Cov. Metrics and Criteria): Given a generalized Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, let $\delta \subseteq \Delta$ be a set of transitions, the weak transition coverage metric for a test suite T on δ is defined as $\frac{|\delta'|}{|\delta|}$, where $\delta' = \{s \rightarrow s' \mid (s \rightarrow s') \in \delta \wedge \exists t \in T.(t \text{ weakly covers } (s \rightarrow s'))\}$. T weakly covers δ if and only if $\delta' = \delta$.

Definition 3.3 (Strong Trans. Cov. Metrics and Criteria): Given a generalized Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$, let $\delta \subseteq \Delta$ be a set of transitions, the strong transition coverage metric for a test suite T on δ is defined as $\frac{|\delta'|}{|\delta|}$, where $\delta' = \{s \rightarrow s' \mid (s \rightarrow s') \in \delta \wedge \exists t \in T.(t \text{ strongly covers } (s \rightarrow s'))\}$. T strongly covers δ if and only if $\delta' = \delta$.

Theorem 3.4 shows that the strong transition coverage criterion subsumes the weak transition coverage criterion.

Theorem 3.4: Given a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$, let $\delta \subseteq \Delta$ be a set of transitions, if a test suite T strongly covers δ, then T also weakly covers δ.

Proof: Since T strongly covers δ, by Definition 3.3, for every $(s \to s') \in \delta$, there exists a t such that (i) t satisfies B; and (ii) for every B's successful run ρ such that $t \vdash \rho$, ss' is a substring of ρ. By (i) and (ii), there exists at least one ρ such that $t \vdash \rho$ and ss' is a substring of ρ. Therefore, by Definition 3.3, T also weakly covers $\mathcal{S}$. □

It shall be noted that covering transitions of a GBA is very different from covering transitions of a Finite State Machine (FSM). Although we use Kripke structure, which is essentially a FSM, to model a system, the criteria defined above focus on covering a property in the form of Büchi automata. By doing so, we are able to measure the semantic adherence of a system with respect to the property, instead of examining only the structure of the system. While focusing on GBA, we are able to reason and test subtle temporal behaviors of the system that cannot be captured by a FSM.

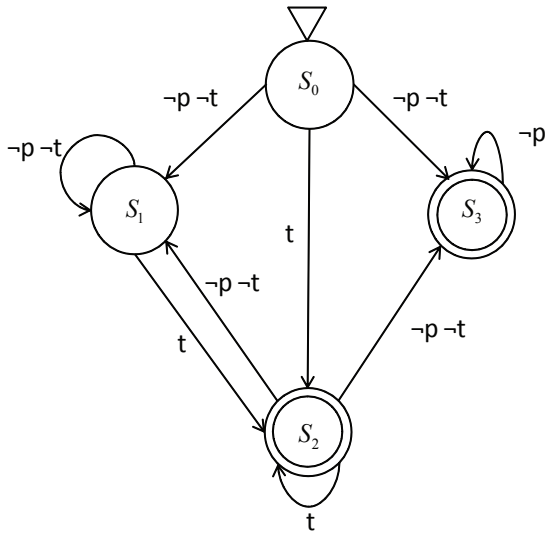

Fig. 2. A general Büchi automaton representing the LTL property $\mathbf{G}(\neg t \Rightarrow ((\neg p \ \mathbf{U} \ t) \vee \mathbf{G}\neg p))$.

IV. Model-Checking-Assisted Test Generation for Transition Coverage

Our model-checking-assisted test case generation algorithms use a Büchi-automaton-based model checker as their underlying engine for searching test cases under a test criterion. Model checking via Büchi automaton is a well-studied subject, and efficient algorithms have been developed over the years (cf. [5]). A Büchi-automaton-based linear temporal model checker such as SPIN [17] verifies a system K against a property ϕ in two stages: first, it constructs a Büchi automaton $B_{\neg\phi}$ for $\neg\phi$; Second, it checks the emptiness of the product of $B_{\neg\phi}$ and K. If the product automaton is empty, then $K \not\models B_{\neg\phi}$, that is, K satisfies ϕ. It shall be noted that although we use SPIN in an implementation of our test-case generation algorithms, these algorithms are not tied to any particular model checker. In fact, the algorithms designate a routine $MC\_isEmpty$ as an abstraction of the core emptiness checking algorithm deployed by many Büchi-automaton-based model checkers. $MC\_isEmpty(B, K)$ checks the emptiness of the product of a Büchi automaton B and a Kripke structure K. $\overline{B}$ accepts K if the result is positive; Otherwise, $MC\_isEmpty(B, K)$ returns a trace α of K that satisfies B. α may be mapped to a word $t = \mathcal{V}\alpha$, which specifies a test case. In model-checking-assisted test case generation, a Büchi automaton used as the input to $MC\_isEmpty(B, K)$ encodes a test objective for a test criterion, and the trace that it returns may be used to construct a test case for that test objective.

The central question in model-checking-assisted test case generation is how to encode a test criterion as temporal properties accepted by a model checker. By Definitions 3.2 and 3.3, a transition coverage criterion may be translated to a set of test objectives, each of which covers a specific transition. Each test objective is captured by a "trap" property in Büchi automaton. For weak transition coverage, a trap property covering a transition $s \to s'$ is a *transition marking general Büchi automaton* (TM-GBA) for $s \to s'$ (Definition 4.1). For strong transition coverage, a trap property covering the transition $s \to s'$ is the negation of a *transition excluding general Büchi automaton* (TE-GBA) for $s \to s'$ (Definition 4.3). Our test generation algorithms used these automata as trap properties to generate test cases.

For a Büchi automaton B and a transition $s \to s'$ of B, the corresponding TM-GBA $B(s \to s')$ keeps two copies of B, and it indexes the states of these copies with 0 and 1, respectively. Two copies are linked by the transition $\langle s, 0\rangle \to \langle s', 1\rangle$. The first copy keeps the start states of B and the second copy keeps the acceptance states. Therefore, a successful run of $B(s \to s')$ must travel from a start state of the first copy to some acceptance states in the second copy, and the only way to do so is to go through the bridging transition $\langle s, 0\rangle \to \langle s', 1\rangle$. By the construction of the TM-GBA $B(s \to s')$, a trace τ can be accepted by $B(s \to s')$ if and only if τ can be accepted by B and it weakly covers $s \to s'$. The formal definition of TM-GBA is presented in 4.1.

Definition 4.1 (TM-GBA): Let $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$ be a GBA, B's transition marking general Büchi automaton $B(s \to s')$ has form of $B(s \to s') = \langle P, S \times \{0,1\}, S_0 \times \{0\}, \Delta', \mathcal{L}', \mathcal{F}' \rangle$, where,

- $\Delta' = \Big(\bigcup_{(s \to s') \in \Delta} \{\langle s, 0\rangle \to \langle s', 0\rangle, \langle s, 1\rangle \to \langle s', 1\rangle\}\Big) \cup \{\langle s, 0\rangle \to \langle s', 1\rangle\}$;
- For every $s \in S$, $\mathcal{L}'((s, 0)) = \mathcal{L}'((s, 1)) = \mathcal{L}(s)$;
- $\mathcal{F}' = \bigcup_{F \in \mathcal{F}} \{F \times \{1\}\}$.

We use an example from our experiments to illustrate the construction. Figure 2 is a generalized Büchi automaton that is semantically equivalent to the LTL property: $\mathbf{G}(\neg t \Rightarrow ((\neg p \ \mathbf{U} \ t) \vee \mathbf{G}\neg p))$. This property specifies a temporal requirement for the GIOP model, the general Inter-Object Request Brokers (ORB) Protocol [18]. Briefly speaking, t stands for a request being sent and p stands for an agent receiving a reply in this formula. Semantically, the formula holds only

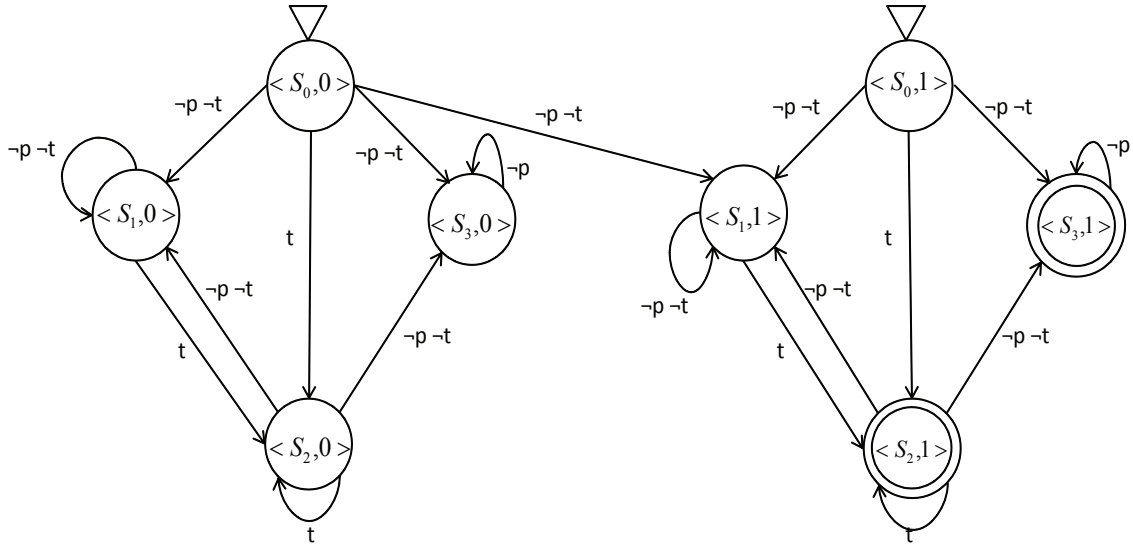

Fig. 3. A transition marking general Büchi automaton covering the transition $s_0 \rightarrow s_1$ for the GBA in Figure 2.

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

Require: B is GBA and K_m is a system model
Ensure: Return the test suite ts that weakly covers all the transitions of B and K_m passes ts. Return $\emptyset$ if such a test suite is not found;

```
for every s → s' ∈ Δ do
    Construct a TM-GBA B(s → s') from B that marks the transition s → s';
    τ = MC_isEmpty(B(s → s'), K_m);
    if |τ| ≠ 0 then
        ts = ts ∪ {V(τ)}
    else
        return ∅;
    end if
end for
return ts;
```

when an agent would not receive any reply until a request has been made. Figure 3 shows a TM-GBA covering the transition $s_0 \rightarrow s_1$ for the GBA in Figure 2. Essentially, the weak transition coverage criterion defines a plain coverage of all the transitions in a GBA, since any test that can induce one successful run will suffice to cover the transition.

Based upon the weak transition coverage criterion, Algorithm 1 generates a test suite that weakly covers all the transitions of a given GBA B. For every transition $s \rightarrow s'$ of the GBA, the algorithm builds a TM-GBA $B(s \rightarrow s')$. The algorithm then calls the emptiness checking routine $MC\_isEmpty$ on $B(s \rightarrow s')$ and a system model K_m. If the product of $B(s \rightarrow s')$ and the system model K_m is not empty, $MC\_isEmpty$ returns a successful run of the product of $B(s \rightarrow s')$ and the system model K_m. A test case related to this successful run is then added to the resulting test suite. Theorem 4.2 shows the correctness of Algorithm 1.

Theorem 4.2: If the test suite ts returned by Algorithm 1 is not empty, then (i) K_m passes ts and (ii) ts weakly covers all the transitions of B.

Proof: (i) For each $t \in ts$, there is a related transition $s \rightarrow s'$ and $MC\_isEmpty(B(s \rightarrow s'), K_m)$ returns a successful run of the production of $B(s \rightarrow s')$ and K_m such that $\mathcal{V}(\tau) = t$. Since any successful run of the production of $B(s \rightarrow s')$ and K_m shall also be a trace of K_m, τ is also a trace of K_m. Therefore, K_m shall pass t. That is, K_m passes every test case in ts.

(ii) As shown in (i), for each $t \in ts$, there is a related transition $s \rightarrow s'$ and a successful run τ of the production of $B(s \rightarrow s')$ and K_m such that $\mathcal{V}(\tau) = t$. We will show that t weakly covers $s \rightarrow s'$. By Definition 3.2, we need to show that there is a successful τ' of B such that τ' takes the transition $s \rightarrow s'$. We obtain the τ' by taking the projection of τ on the states of B as follows: since τ is a run of the production of the TM-GBA $B(s \rightarrow s')$ and Kripke structure K_m, each state on τ has the form of $\langle\langle s', i\rangle, v\rangle$, where s' is a state of B, $i \in \{0, 1\}$ is an index number marking the states in the TM-GBA $B(s \rightarrow s')$, and v is a state of K_m. We project the state $\langle\langle s', i\rangle, v\rangle$ to the state s' on B. Let τ' be the resulting sequence. Clearly τ' is also a successful run of B because, by Definition 4.1, each transition in $B(s \rightarrow s')$ is mapped to a transition in B and each acceptance state in $B(s \rightarrow s')$ is mapped to an acceptance state in B. In addition, τ has to go through $\langle s, 0\rangle \rightarrow \langle s', 1\rangle$ because, by Definition 4.1, acceptance states of a TM-GBA are indexed by 1, whereas start states are indexed by 0, and hence the only way that a run of $B(s \rightarrow s')$ is successful is that it has to go through $\langle s, 0\rangle \rightarrow \langle s', 1\rangle$. Therefore, τ' has to take $s \rightarrow s'$, and we proved (ii). □

Followed from their definitions, strong transition coverage criterion subsumes its *weak* counterpart. To generate test cases that strongly cover transitions in a Büchi automaton, we need to construct a transition excluding generalized Büchi automaton (TE-GBA). For a Büchi automaton B and a transition

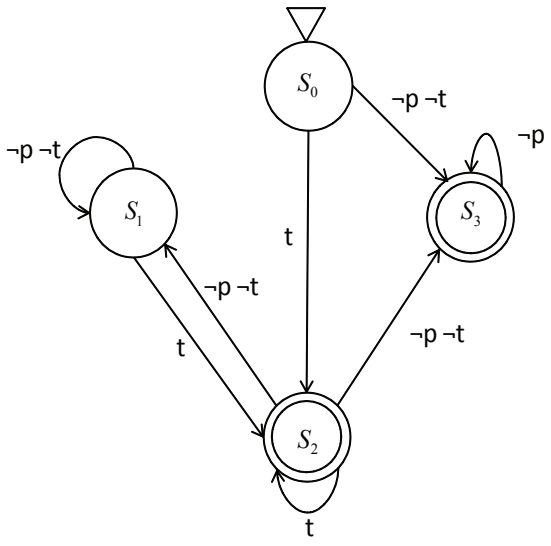

Fig. 4. A transition excluding general Büchi automaton covering the transition $s_0 \to s_1$ for the GBA in Figure 2.

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

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 all the transitions of B and K_m passes ts. Return $\emptyset$ if such a test suite is not found;

```
for every (s → s') ∈ Δ do
    B_{¬(s→s')} = ⟨P, S, S_0, Δ − {s → s'}, L, F⟩;
    τ = MC_isEmpty(B_{¬(s→s')}, K_m));
    if |τ| ≠ 0 then
        ts = ts ∪ {V(τ)}
    else
        return ∅;
    end if
end for
return ts;
```

$s \to s'$ of B, the TE-GBA $B_{\overline{s\to s'}}$ is obtained by simply removing $s \to s'$ from B. In this case, for every trace τ accepted by B, τ does not satisfy $B_{\overline{s\to s'}}$, i.e., $\tau \models \overline{B_{\overline{s\to s'}}}$, the compliment Büchi automaton of TE-GBA if and only if τ strongly covers $s \to s'$. Figure 4 shows a TE-GBA covering the transition $s_0 \to s_1$ for the GBA in Figure 2.

Definition 4.3 (TE-GBA): Let $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F}\rangle$ be a GBA, a transition excluding generalized Büchi automaton for $s \in S$ is a GBA $B_{\overline{s\to s'}} = \langle P, S, S_0, \Delta - \{s \to s'\}, \mathcal{L}, \mathcal{F}\rangle$.

Algorithm 2 generates a test suite that strongly covers the transitions of a given GBA B. For every transition of the GBA, the algorithm builds a TE-GBA. The algorithm then calls the emptiness checking routine $MC\_isEmpty$ on the negation of the TE-GBA and the system model. If the product of $\overline{B_{\overline{s\to s'}}}$ and the system model K_m is not empty, the function $MC\_isEmpty$ returns a successful run of the product of $\overline{B_{\overline{s\to s'}}}$ and the system model K_m. Since this run is also a trace of K_m, it can be further mapped to a test case consisting of a series of atomic propositions. The test case is then added to the resulting test suite. Theorem 4.4 shows the correctness of Algorithm 2.

Theorem 4.4: If the test suite ts returned by Algorithm 2 is not empty, then (i) K_m passes ts; and (ii) ts strongly covers all the transitions of B that can be strongly covered.

Proof: (i) For each $t \in ts$, there is a related transition $s \to s'$ and $MC\_isEmpty((\overline{B_{\overline{s\to s'}}}), K_m)$ returns a successful run of the product of $\overline{B_{\overline{s\to s'}}}$ and K_m such that $\mathcal{V}(\tau) = t$. Since any successful run of the production of $\overline{B_{\overline{s\to s'}}}$ and K_m is also a successful run on K_m, and K_m shall pass $t = \mathcal{V}(\tau)$. Therefore, K_m passes every test case in ts.

(ii) As shown in (i), for each $t \in ts$, there is a related transition $s \to s'$ and a successful run τ of the production of $\overline{B_{\overline{s\to s'}}}$ and K_m such that $\mathcal{V}(\tau) = t$. We will show that t strongly covers the transition $s \to s'$.

First, since τ is also a trace of K_m and K_m satisfies B by the precondition of Algorithm 2, $\tau \models B$.

Next, we will prove by contradiction that every successful run of B that is induced by the test case t shall take the transition $s \to s'$ at least once. Suppose not, and let ρ be a successful run of B that is induced by t and ρ does not take $s \to s'$. It follows that ρ shall also be a successful run of $B_{\overline{s\to s'}}$, because the only difference between $B_{\overline{s\to s'}}$ and B is that $B_{\overline{s\to s'}}$ does not have transition $s \to s'$. Therefore, t shall also be accepted by $\overline{B_{\overline{s\to s'}}}$. This contradicts to the fact that t is accepted by $\overline{B_{\overline{s\to s'}}}$, which is a complement of $B_{\overline{s\to s'}}$, and thus should have no common words in their languages. If t can be accepted by both automaton, then $t \in L(\overline{B_{\overline{s\to s'}}}) \cap L(B_{\overline{s\to s'}}) \neq \emptyset$. Therefore, every successful run of B that accepts t shall visit $s \to s'$ at least once. □

A GBA B can be translated to a Büchi automaton (BA) by indexing acceptance states. The resulting BA has the size $O(|\mathcal{F}| \cdot |B|)$, where $|\mathcal{F}|$ is the number of acceptance state sets in B, and $|B|$ is the size of B. The emptiness checking for a BA can be done in linear time (cf. [19]). Therefore, generating a test case weakly covering a transition can be done in $O(|K|\cdot|\mathcal{F}|\cdot|B|)$, where $|K|$ is the size of the model. Generating a test suite weakly covering all the transitions in B can be done in $O(|K|\cdot|\mathcal{F}|\cdot|B|^2)$. Algorithm 2 starts with the construction of a TE-GBA for a transition, which can be done in linear time. It then negates the TE-GBA. Michel [20] provided a lower bound of $2^{O(nlogn)}$ for negating a BA of size n. Therefore, Algorithm 2 takes at least $O(|K| \cdot 2^{O(|\mathcal{F}|\cdot|B|log(|\mathcal{F}|\cdot|\mathcal{B}|)}$ to generate a test case strongly covering a transition and at least $O(|K| \cdot |B| \cdot 2^{O(|\mathcal{F}|\cdot|B|log(|\mathcal{F}|\cdot|\mathcal{B}|)}) = O(|K| \cdot 2^{O(|\mathcal{F}|\cdot|B|log(|\mathcal{F}|\cdot|\mathcal{B}|)})$ to generate a test suite strongly covering all the transitions of a GBA. Generating test cases under strong transition coverage is much more computationally expensive than doing so under weak transiton overage, and the reason can be traced back to Definition 3.1: strongly covering a transition $s \to s'$ requires all the successful runs induced by a test to visit $s \to s'$ at least once, whereas weakly covering s only requires a single successful run visiting $s \to s'$.

V. Transition-Coverage-Induced Requirement Refinement

Specification-based testing checks whether a system conforms to its specification. Insufficient coverage on a specification may be caused by a problem in a system, but it may be also due to the deficiency of the specification. For example, the specification may be imprecise and/or too general. Our model-checking-assisted test case generation algorithms use a model-checker to systematically search the state space of a system for test cases under a transition coverage criterion. We use the information from this process of test case generation to refine the formal specification in Büchi automaton.

In this work we consider the refinement with respect to language inclusion. Formally we define $B' \sqsubseteq B$ if and only if $L(B') \subseteq L(B)$, that is, B' is a refinement of B if the language accepted by B' is a subset of the language accepted by B. By Definition 2.3, one can infer that a test accepted by B' will also be accepted by B. In our transition-coverage-induced requirement refinement, the feedback from model-checking-assisted test case generation is used to refine a temporal property B in Büchi automaton. The resulting property B' will be more semantically restricted than the original property, that is, $B' \sqsubseteq B$.

Lemma 5.1: Given a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$ with a transition $s \to s' \in \Delta$, let $B_{\overline{s \to s'}} = \langle P, S, S_0, \Delta - \{s \to s'\}, \mathcal{L}, \mathcal{F} \rangle$ be the transition excluding GBA for $s \to s'$, then $B_{\overline{s \to s'}} \sqsubseteq B$.

Proof: By Definition 4.3 the TE-GBA $B_{\overline{s \to s'}}$ misses transition $s \to s'$. It follows that the successful runs of $B_{\overline{s \to s'}}$ are those that do not visit $s \to s'$ in the original GBA. Therefore, $L(B_{\overline{s \to s'}}) \subseteq L(B)$ and hence $B_{\overline{s \to s'}} \sqsubseteq B$. □

Theorem 5.2: Given a GBA $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$ with a transition $s \to s' \in \Delta$ and a Kripke structure $K = \langle V, v_0, \to, \mathcal{V} \rangle$, let $B_{\overline{s \to s'}} = \langle P, S, S_0, \Delta - \{s \to s'\}, \mathcal{L}, \mathcal{F} \rangle$ be the transition excluding GBA for s, if K passes a test case t strongly covering $s \to s'$ and $K \models B$, then, $K \not\models B_{\overline{s \to s'}}$.

Proof: We will prove by contradiction. Suppose that $K \models B_{\overline{s \to s'}}$. Since K passes t, K has a trace τ such that $\mathcal{V}(\tau) = t$. Since $K \models B_{\overline{s \to s'}}$, $t \models B_{\overline{s \to s'}}$. Let ρ be a successful run of $B_{\overline{s \to s'}}$ induced by t, that is, $t \vdash \rho$. ρ is also a successful run of B because by Lemma 5.1 $B_{\overline{s \to s'}}$ is a refinement of B. Since ρ does not visit $s \to s'$, t does not strongly covers $s \to s'$, which contradicts to the condition of the theorem. Therefore, $K \not\models B_{\overline{s \to s'}}$. □

Definition 5.3 (Vacuous Transitions): Given a generalized Büchi automaton $B = \langle P, S, S_0, \Delta, \mathcal{L}, \mathcal{F} \rangle$ and a Kripke structure K, a transition $s \to s'$ of B is vacuous with respect to K if and only if $K \models B$ implies $K \models B_{\overline{s \to s'}}$, where $B_{\overline{s \to s'}} = \langle P, S, S_0, \Delta - \{s \to s'\}, \mathcal{L}, \mathcal{F} \rangle$ is the TE-GBA for $s \to s'$.

Since $B_{\overline{s \to s'}}$ is a refinement of B, $K \not\models B$ implies $K \not\models B_{\overline{s \to s'}}$. Therefore, Definition 5.3 essentially states that a vacuous transition $s \to s'$ of a GBA B for a Kripke structure K does not affect whether K satisfies B. That is, if we remove the vacuous transition $s \to s'$ from B, the outcome of whether the system K satisfies GBA B will stay the same. This observation prompts us to introduce the notion of transition-coverage-induced refinement: *for a given system and a property in a GBA, if a transition of the GBA is vacuous to the system, the transition can be removed from the GBA, and the system still satisfies this refinement of the original GBA.*

Corollary 5.4: Given a generalized Büchi automaton B and a Kripke structure $K = \langle V, v_0, \to, \mathcal{V} \rangle$, $s \to s'$ is not vacuous with respect to K if and only if $K \models B$ and there exists a test t such that t strongly covers $s \to s'$ and K passes t.

Proof: Note that $K \not\models B$ implies $K \not\models B_{\overline{s \to s'}}$ since $L(B_{\overline{s \to s'}}) \subseteq L(B)$. By Definition 5.3, $s \to s'$ is not vacuous with respect to K if and only if $K \models B$ and $K \not\models B_{\overline{s \to s'}}$. Therefore, we only need to show that given $K \models B$, $K \not\models B_{\overline{s \to s'}}$ if and only if there exists a test t strongly covering $s \to s'$ and K passes t.

($\Rightarrow$) Since $K \models B$ and $K \not\models B_{\overline{s \to s'}}$, there must be a trace τ of K such that (i) B has a successful run ρ such that $\mathcal{V}(\tau) \vdash \rho$, and (ii) $B_{\overline{s \to s'}}$ does not have a successful run ρ' such that $\mathcal{V}(\tau) \vdash \rho'$. Since the TE-GBA $B_{\overline{s \to s'}}$ is obtained by removing transition $s \to s'$ from B, it follows that B's every successful run ρ'' such that $\mathcal{V}(\tau) \vdash \rho''$ shall go through $s \to s'$, otherwise, ρ'' is also a successful run of $B_{\overline{s \to s'}}$, which contradicts to the condition (ii) for τ. Now let $t = \mathcal{V}(\tau)$. By Definition 3.1, t strongly covers s and K passes t.

($\Leftarrow$) Since K passes t, K has a trace τ such that $\mathcal{V}(\tau) = t$. Since t strongly covers $s \to s'$, we have (i) $t \models B$, and hence B has a successful run ρ such that $t \vdash \rho$; and (ii) for every successful run ρ' of B such that $t \vdash \rho'$, ρ' goes through $s \to s'$. We will prove by contradiction that $K \not\models B_{\overline{s \to s'}}$. Suppose that $K \models B_{\overline{s \to s'}}$. It follows that every trace of K shall be accepted by $B_{\overline{s \to s'}}$, and hence $B_{\overline{s \to s'}}$ has a successful run ρ'' such that $\mathcal{V}(\tau) \vdash \rho''$. Note that $B_{\overline{s \to s'}}$ is obtained by removing $s \to s'$ from B, ρ'' is also a successful run of B but ρ'' does not visit $s \to s'$. It follows that t cannot strongly cover s because B has a successful run induced by t that does not visit $s \to s'$. We reach a contradiction. Therefore, $K \not\models B_{\overline{s \to s'}}$. □

Corollary 5.4 shows the relation between the strong transition coverage for a transition of a GBA and its non-vacuousness. It shall be noted that testing alone cannot prove the non-vacuousness of a transition of a GBA. This is because the non-vacuousness of a transition $s \to s'$ of B for a system K requires that $s \to s'$ impacts in some way the outcome of whether K satisfies B, that is, either $K \models B$ and $K \not\models B_{\overline{s \to s'}}$, or $K \not\models B$ and $K \models B_{\overline{s \to s'}}$. Since $B_{\overline{s \to s'}}$ is obtained by removing $s \to s'$ from B, $K \not\models B$ implies $K \not\models B_{\overline{s \to s'}}$. The only possibility left is that $K \models B$ and $K \not\models B_{\overline{s \to s'}}$, but testing alone may not conclusively prove that K satisfies B. Nevertheless, lack of the strong coverage for transition $s \to s'$ indicates that $s \to s'$ is a vacuous transition for K. Therefore, we can remove $s \to s'$ from B without affecting the outcome of whether K satisfies B.

Algorithm 3 refines a GBA while generating a test suite strongly covering the transitions of the new GBA. Algorithm 3 is a modification of Algorithm 2. The difference is line 7. Instead of returning with a failed attempt in Algorithm 2 when the full strong state coverage cannot be achieved, Algorithm 3

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

.

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

Ensure: Return a GBA as a refinement of B, and a test suite ts that strongly covers all the transitions of the new GBA;

1: **for** every $s \rightarrow s' \in S$ **do**
2: $B_{\overline{s \rightarrow s'}} = \langle P, S, S_0, \Delta - \{s \rightarrow s'\}, \mathcal{L}, \mathcal{F} \rangle$;
3: $\tau = MC\_isEmpty(\overline{B_{\overline{s \rightarrow s'}}}, K_m))$;
4: **if** $|\tau| \neq 0$ **then**
5: $ts = ts \cup \{\mathcal{V}(\tau)\}$
6: **else**
7: $B = B_{\overline{s \rightarrow s'}}$;
8: **end if**
9: **end for**
10: **return** $\langle B, ts \rangle$;

refines the input GBA by removing vacuous transitions. The output will be a GBA refined by removing vacuous transitions with respect to the model, and a test suite for the refined GBA.

VI. Experiment

To evaluate the effectiveness of our transition coverage criteria, we compare them against other criteria using cross-coverage measurement. We conduct the experiment on examples from a variety of fields. The first example is from software engineering. It tests the criteria on a model of the general Inter-ORB Protocol (GIOP), a key component of the Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) specification [18]. The second example is from computer networks. It tests the criteria on a model of a sliding window protocol, which depicts the behavior of the classic network protocol as in [21]. The third example is from computer security. It tests the criteria on a model for the Needham-Schroeder security protocol as described in [22]. The last two examples are from operating systems. They test the criteria on Lamport's Bakery algorithm [23] and Peterson's algorithm [24] for mutual exclusion problem, respectively. The properties we used in our study all model the central requirements of the systems. For the GIOP model, the property models the requirement on the behaviors of the recipient agent during communication. The LTL property for the sliding window checks the correctness of transmission among multiple channels. The property for Needham-Shroeder security protocol is a liveness property requiring that the initiator can only send messages after the responder is up and running. Finally, the properties for the two mutual exclusion problem require the 3 critical requirements to hold: mutual exclusion, progress, and bounded waiting.

Table I provides an overview of the models and properties, showing the size of both the models and properties in terms of the number of branches, clauses, states/transitions of the underlying Büchi automata, atomic propositions in the properties, and definition-usage pairs. All of the information in Table I are of relevance to the diversified profiles of test criteria we used in the experiment for the comparison, in terms of the size of test suites generated.

TABLE I. Overview of the models and properties used in the experiments.

Models	Branches	Clauses	States	Transitions	Atoms.	D-U pairs
GIOP	77	77	3	9	4	256
Slid. Win.	24	31	4	11	4	83
Needham	41	50	3	9	3	120
Lamport's	17	20	3	16	5	77
Peterson	5	8	3	16	5	30

In each set of our experiment, a test suite is generated for each criterion (listed at the leftmost column in Table II). We measure the coverage of the test suite under other criteria (listed at the top rows in Table II). Branch coverage (BC) and clause coverage (CC) are two variations of the logic expression criteria [25] and among the most commonly-used structural test criteria. Branch coverage criterion evaluates the number of branches being covered, whereas the clause coverage criterion examines the truth value of each clause within each branch. We include a strong state coverage criterion (SC/strong) and a weak state coverage criterion (SC/weak) for Büchi automaton, defined in [1]. We also include a property-coverage criterion (PC) for Linear Temporal Logic (LTL), defined in [12]. In addition, we include the data-flow coverage criterion (DC) defined in [26]. Finally, we compare the proposed strong transition coverage criterion (TC/strong) and its weak variant (TC/strong) with each other, as well as with other criteria.

In [27] we developed a uniform framework and tool to compare the effectiveness of test criteria used with model-checking-assisted test case generation. This experiment uses an extension of the tool that also supports transition coverage criteria proposed in this paper. We use GOAL [28] to perform graph transformation required for building TE-GBAs and TM-GBAs. We use SPIN [17] as the underlying model checker to assist test case generation.

Table II shows the results of our experiment. Numbers in a parenthesis represents the coverage measured by a test criterion for a test suite generated for the same criterion. A less-than perfect coverage in this case may indicate the deficiency of a model and/or a requirement. For instance, for the sliding window example, the test suite for the strong transition coverage criterion may only reach 67% coverage due to the existence of vacuous transitions in the specification.

The overall result indicates that the transition coverage criteria in general have a more competent performance over the others, especially when compared with traditional structural criteria such as branch, clause, and data-flow coverage criteria. For instance, the test suite generated for the strong transition coverage achieves full coverage over these criteria in Lamport's Bakery algorithm for mutual exclusion, whereas the test suites generated for branch, clause, and data-flow coverage only reach 33%, 56%, and 33%, respectively. Other three property-based coverage criteria - strong and weak state coverage criteria for Büchi automaton and the property-coverage criterion - also exhibit a strong performance, albeit stop short of those of transition coverage criteria.

The results also indicate that for the GIOP and sliding

TABLE II. CROSS-COVERAGE COMPARISON RESULTS

	GIOP							
	BC	CC	SC/strong	SC/weak	TC/strong	TC/weak	PC	DC
BC	(100%)	100%	67%	67%	56%	56%	75%	100%
CC	100%	(100%)	67%	67%	56%	56%	75%	100%
SC/strong	73%	73%	(100%)	100%	100%	100%	75%	82%
SC/weak	77%	77%	100%	(100%)	100%	100%	75%	82%
TC/strong	77%	77%	100%	100%	(100%)	100%	75%	82%
TC/weak	77%	77%	100%	100%	100%	(100%)	75%	82%
PC	73%	73%	100%	100%	77%	77%	(75%)	78%
DC	71%	71%	100%	100%	77%	77%	75%	(100%)
	Sliding Window							
	BC	CC	SC/strong	SC/weak	TC/strong	TC/weak	PC	DC
BC	(100%)	93%	50%	75%	50%	67%	75%	61%
CC	100%	(96%)	50%	75%	50%	67%	50%	51%
SC/strong	67%	81%	(75%)	75%	42%	67%	75%	70%
SC/weak	67%	81%	75%	(75%)	42%	67%	75%	70%
TC/strong	72%	87%	75%	75%	(67%)	67%	75%	70%
TC/weak	72%	87%	75%	75%	67%	(83%)	75%	83%
PC	72%	87%	75%	75%	67%	75%	(75%)	61%
DC	100%	93%	75%	75%	42%	67%	75%	(100%)
	Needham Protocol							
	BC	CC	SC/strong	SC/weak	TC/strong	TC/weak	PC	DC
BC	(100%)	100%	67%	100%	77%	100%	67%	100%
CC	100%	(100%)	67%	100%	77%	100%	67%	100%
SC/strong	47%	47%	(100%)	100%	77%	77%	100%	41%
SC/weak	25%	25%	100%	(100%)	67%	67%	100%	20%
TC/strong	51%	51%	100%	100%	(77%)	100%	100%	44%
TC/weak	45%	45%	100%	100%	77%	(100%)	100%	40%
PC	51%	51%	100%	100%	55%	55%	(100%)	41%
DC	100%	100%	67%	100%	77%	100%	67%	(100%)
	Lamport's Bakery							
	BC	CC	SC/strong	SC/weak	TC/strong	TC/weak	PC	DC
BC	(100%)	100%	33%	100%	33%	100%	60%	70%
CC	100%	(100%)	67%	67%	56%	56%	75%	100%
SC/strong	100%	100%	(100%)	100%	69%	100%	100%	100%
SC/weak	100%	100%	100%	(100%)	69%	100%	100%	100%
TC/strong	100%	100%	100%	100%	(88%)	100%	100%	100%
TC/weak	100%	100%	100%	100%	88%	(100%)	100%	100%
PC	75%	70%	100%	100%	75%	100%	(100%)	81%
DC	100%	100%	33%	100%	33%	100%	100%	(100%)
	Peterson							
	BC	CC	SC/strong	SC/weak	TC/strong	TC/weak	PC	DC
BC	(100%)	85%	67%	67%	56%	56%	60%	100%
CC	100%	(100%)	67%	67%	56%	56%	75%	100%
SC/strong	100%	85%	(100%)	100%	63%	100%	100%	100%
SC/weak	100%	85%	100%	(100%)	63%	100%	100%	100%
TC/strong	100%	90%	100%	100%	(88%)	100%	100%	100%
TC/weak	100%	90%	100%	100%	88%	(100%)	100%	100%
PC	100%	100%	100%	100%	63%	100%	(100%)	100%
DC	100%	85%	67%	67%	56%	56%	80%	(100%)

window models, transition coverage criteria and the other three property-based criteria are not able to achieve a perfect coverage over the branches and data-flow paths. The reason behind this is that the properties used in the experiment do not fully cover all the functional aspects of these models. For instance, in the GIOP example, the only property used in the experiment specifies the requirement for the recipient's behavior when it is waiting for or receives a message. It does not concern the other functions of the model. Therefore the test suite generated for the property bypasses some code segments, which leads to a less-than perfect structural coverage.

This observation suggests an important trait of property-based coverage criteria, including the transition coverage criteria proposed in this paper: the performance of these criteria are largely influenced by the quality of a temporal requirement. A well-defined requirement that touches more aspects of a model may result in better coverage, whereas the deficiency in the requirement can lead to a lower coverage on the system. For the same reason, the results from property-based coverage testing may be used to identify and correct the deficiency in the specification and the system model. This potential benefit of property-based coverage testing is capitalized by our transition-coverage-induced property refinement in Section V.

Another point worth noting is the relation between the transition coverage criteria and state coverage criteria. It is straightforward to prove that a transition coverage criterion subsumes its state coverage counterpart. If a transition $s \to s'$ is strongly (or, weakly) covered by a test case t, both s and s' have to be strongly (or, weakly) covered by t. Practically, an automaton generally has more transitions than states, thus a test suite generated for a transition coverage criterion is able to cover more scenarios than the test suite generated for its state

counterpart. On the other hand, it is also more computationally expensive to generate a test suite for a transition coverage criterion than for its state coverage counterpart.

VII. Conclusions

We considered specification-based testing for linear temporal properties expressed in generalized Büchi automata (GBAs). We introduced two variants of transition coverage metrics and criteria for measuring how well the transitions of a specification in a GBA are covered during a test. The immediate application of these two metrics and criteria is to select test cases based on their relevancy to a GBA-based specification. For this application we provided model-checking-assisted algorithms to automate test case generation for proposed coverage criteria. Our strategy, while bearing a similarity to transition coverage of a FSM, is fundamentally different in several aspects.

First of all, we focus on the properties, instead of the systems. By doing so, we are able to examine the degree of adherence between a system and its requirements. Secondly, with the benefit of the power of both GBA and model checkers, we can explore complicated behaviors of the system, hence making the testing much more thoroughly. Last but not least, this work extends our previous work on state coverage metrics for Büchi automaton [1] and vacuity-based coverage metric for LTL formulae [12]. Property-based metrics such as these proposed in this paper also help identify the deficiency in the requirement specification. We showed that the feedback from model-checking-assisted test case generation for the proposed transition coverage criteria might also be used to refine the requirement specification. We defined the notion of vacuous transitions, and the removal of these vacuous transitions refines the requirement specification. The result is a refined requirement that more closely describes the behaviors of a system. We also developed a model-checking-assisted approach to formalize and automate the process of requirement refinement. We tested the proposed transition coverage metrics and criteria on a variety of applications, and the results showed their effectiveness.

This research represents an important step in our effort to harness the synergy between model checking and testing for making tests more efficient and effective. For future work, we will continue to explore syntactical as well as semantic coverage metrics for specification-based testing with formal temporal properties.

References

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

[2] S.-. Committee, "Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, Tech. Rep., 1992.

[3] M. Heimdahl, S. Rayadurgam, and W. Visser, "Specification centered testing," in *Proceedings of the Second International Workshop on Automated Program Analysis, Testing and Verification*, 2001.

[4] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[5] R. Gerth, D. Peled, M. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *PSTV'95*. Chapman and Hall, 1995, pp. 3–18.

[6] D. Giannakopoulou and F. Lerda, "From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata," in *Lecture Notes In Computer Science; Vol. 2529*. Springer-Verlag, 2002.

[7] G. J. Holzmann, "The Model Checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.

[8] P. Ammann and P. E. Black, "A specification-based coverage metric to evaluate test sets," in *The 4th IEEE International Symposium on High-Assurance Systems Engineering*, ser. HASE '99. Washington, DC, USA: IEEE Computer Society, 1999.

[9] G. Fraser and A. Gargantini, "An Evaluation of Specification Based Test Generation Techniques Using Model Checkers," in *2009 Testing: Academic and Industrial Conference - Practice and Research Techniques*. Ieee, 2009.

[10] A. Calvagna and A. Gargantini, "A Logic-Based Approach to Combinatorial Testing with Constraints," in *2nd International conference on Tests and Proofs*, 2008.

[11] H. S. Hong, I. Lee, O. Sokolsky, and H. Ural, "A temporal logic based theory of test coverage and generation," in *TACAS'02*, ser. LNCS, vol. 2280, 2002, pp. 327–341.

[12] L. Tan, O. Sokolsky, and I. Lee, "Specification-based Testing with Linear Temporal Logic," in *IRI'04*. IEEE society, 2004, pp. 493–498.

[13] O. Kupferman and M. Y. Vardi, "Vacuity detection in temporal model checking," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 4, no. 2, Feb. 2003.

[14] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller, "Coverage metrics for requirements-based testing," in *Proceedings of the 2006 international symposium on Software testing and analysis*, ser. ISSTA '06. New York, NY, USA: ACM, 2006.

[15] A. Rajan, "Coverage metrics for requirements-based testing," Ph.D. dissertation, University of Minnesota, Minneapolis, MN, USA, 2009.

[16] 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.

[17] G. J. Holzmann, "The model checker SPIN," *IEEE Trans. Softw. Eng.*, vol. 23, May 1997.

[18] 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.

[19] M. Vardi, "Automata-theoretic model checking revisited," in *8th Verification, Model Checking, and Abstract Interpretation*. Springer, 2007.

[20] M. Michel, *Complementation is more difficult with automata on infinite words*. Paris, France: CNET, 1988.

[21] Andrew S. Tanenbaum, *Computer Networks*, 5th ed. Prentice Hall, 2010.

[22] P. Maggi and R. Sisto, "Using spin to verify security properties of cryptographic protocols," in *Proceedings of the 9th International SPIN Workshop on Model Checking of Software*. London, UK, UK: Springer-Verlag, 2002.

[23] L. Lamport, "A new solution of dijkstra's concurrent programming problem," *Commun. ACM*, vol. 17, August 1974.

[24] G. L. Peterson, "Myths about the mutual exclusion problem," *Inf. Process. Lett.*, vol. 12, no. 3, 1981.

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

[26] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. Softw. Eng.*, vol. 11, April 1985.

[27] 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.

[28] 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.