Testing with Büchi Automata: Transition Coverage Metrics, Performance Analysis, 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 one of the most commonly used formalisms for specifying and reasoning linear temporal properties. It is instrumental for developing formal verification algorithms, i.e., model checkers, for linear temporal logics. Until now Büchi automaton-based specification is mainly used in automated verification in form of linear temporal logic model checking. In this paper, we develop test criteria and techniques that are essential for testing upon specifications in Büchi automata. These criteria measure the semantic relevancy of test cases to a requirement in Büchi automaton. We define "weak" and "strong" variants of the criteria based on coverage on the transitions of the Büchi automaton. These criteria can be used to measure the quality of existing test cases with respect to requirements in Büchi automaton, and to drive test case generation. We develop automated test-case generation algorithms that use an off-the-shelf model checker to generate test cases under these test criteria. In our extended computational study we deploy two methodologies to measure and demostrate the effectiveness of our approach. First, we measure the cross-coverage of the transition coverage criteria against other existing test criteria. Second, we use a fault-injection technique to measure the sensitivity of our approach. In both cases, our approach shows a better performance compared with existing test criteria and a good sensitivity in detecting errors systematically injected to a system. Furthermore, the proposed criteria uncover not only the deficiency of a test suite with respect to a linear temporal requirement, but also that of the requirement itself. We propose an algorithm to refine the requirement using the feedback from test-case generation.

1 Introduction

Verification and validation (V&V) is an essential activity in software engineering. With our society increasingly replying on software, from embedded software in automobile, to mobile apps, and anything between, the expectation for the quality of software and the cost of V&V activities are also soaring. Heimdahl

* Correspondent Author

et. al [12] estimated that V&V activities take up around 50%-70% of resource in the development of high-dependable software. To meet the expectation for the quality and to reduce the cost, an organization often develops its own V&V strategy which deploys a mixture of V&V techniques. Two most commonly used V&V techniques are testing and formal verification.

Testing is a classical V&V technique, and its origin may be dated back to the onset of modern engineering activities. Testing checks the behaviors of a system under controlled input stimuli, also known as test cases. Over years, testing has been incorporated into various software quality standards. For example, DO-178 [5] is the software quality standard for safety-critical avionic software. A key component of DO-178 is a set of required structural testing criteria, including the MC/DC criterion [3]. Among the strengths of testing are its scalability and versatility: generally it scales well for a large system, and it may be used for testing a high-level model of a system, such as in the case of model-driven testing [1], and the system implementation. The major drawback of testing is that it can only show the presence, but not absence of bugs, as famously noted by Dijkstra [6].

Formal verification, on the other hand, refers to an array of techniques that build a mathematically rigid proof for the correctness of a system design with respect to its specifications. One of the most commonly used formal verification techniques is model checking [4], in which a specification of a system is encoded in a temporal logic. The strenghes and weaknesses of formal verification are often complemental to those of testing: formal verification may be used to establish the correctness of the correctness of a system design on a mathematically sound ground. Nevertheless, it is limited on its scalability and it is generally only applicable to a design, not an implementation of a system.

Our research aims to harness the synergy of these two mostly commonly used and yet complementary techniques, testing and formal verification, to build more efficient and effective V&V processes. With the rising popularity of formal verification techniques, particularly, model checking, requirements are increasingly specified in a formalism that facilitates formal verification. An emerging research theme is *how testing may take advantage of formal requirements initially intended for model checking*. In this paper we address the issue in the context of specification-based testing with Büchi automaton. As a form of ω-automata, Büchi automaton accepts ω-language, an extension of regular languages with infinite words. Büchi automaton has been an instrumental tool in linear temporal logic model checking [9,11]. It has been used for specifying linear-temporal properties directly. Büchi automaton also serves as an unified intermediate format into which a requirement in other linear temporal logics such as Linear Temporal Logic may be translated, before the requirement is processed by a model checker.

We develop a set of techniques to meet the challenges of specification-based testing with Büchi automata. The core of our approach is a set of test criteria that measure how relevant a test case with respect to a requirement encoded in a Büchi automaton. Specifically our coverage criteria require a test suite to

cover transitions of a Büchi automaton. We define "weak" and "strong" variants of these criteria, which define different degree of relevancy, to reflect the nondeterministic nature of a Büchi automaton.

To improve the efficiency of specification-based testing, we develop a model-checking-assisted algorithm for generating test cases under the proposed criteria. Utilizes the counterexample generation capability provided by an existing linear-temporal model checker, the algorithm automates the test-case generation for a system and its requirement in Büchi automaton. Our test-case generation algorithm works as a two-step process: first, it synthesizes a property for each transition of the Büchi automaton. Also known as a "trap" property, the property characterizes a test case covering the transition of the Büchi automaton. Next, the algorithm uses the counter-example mechanism of an off-the-shelf model checker to generate the test case satisfying the "trap" property. In our experiment, we build a model-checking-assisted test-case generator using the model checker SPIN [13].

The proposed test criteria and metrics not only measure the quality of a test suite in terms of its relevancy to a requirement in Büchi automaton, they may also be used to identify the deficiency of the requirement itself. Besides insufficient test suite, incorrect or imprecise requirements may cause lack of transition coverage for a Büchi automaton. By utilizing the feedback from our model-checking-assisted test generation algorithm, we are able to identify the deficiency of a requirement. By extending the model-checking-assisted test generation algorithm, we develop an algorithm to refine the requirement in Büchi automaton.

To better assess the performance of our approach, we carry an extended computational study using two different methodologies. The first method is to measure the cross coverage between our test criteria and several popular existing criteria, including data-flow criteria and branch coverage criterion, as well as property-coverage criteria that we proposed before. The cross coverage measures how much a test suite generated under one criterion covers another criterion; the second method is to measure the effectiveness of different test criteria in catching errors systematically introduced (i.e. "injected") to a system. In both cases, our approach shows a better cross coverage and an improved effectiveness in catching bugs, especially when considering the cost involving in performing tests.

The rest of the paper is organized as follows: Section 2 prepares the notations used in the paper; Section 3 introduces both variants of transition coverage metrics and criteria for Büchi automata; Section 4 presents the model-checking-assisted test case generation algorithms for the transition coverage criteria; Section 5 discusses the requirement refinement using the feedback from the model-checking-assisted test case generation; Section 6 discusses the result of our computational study on the performance comparison between the new criteria and other existing test criteria, using cross-coverage comparison and fault-injection-based sensitivity analysis; and finally Section 7 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. A central question in model-checking-assisted test generation is how to specify the test objectives as temporal properties acceptable by a model checker. Fraser and Gargantini [7] showed that traditional structural coverage criteria such as MC/DC Coverage could be expressed in Computational Tree Logic (CTL), which were accepted by a CTL model checker such as NuSMV for test generation. Hong *etal.* also used CTL to translate the data flow criteria [14]. Calvagna and Gargantini encoded several combinatorial testing criteria in Linear Temporal Logic (LTL) for the model checker SAL for similar purposes [2]. These previous works applied the model-checking-assisted on existing testing criteria. In contrast, we studied test generation with temporal logic requirements in [23]. Inspired by the notion of vacuity in [18], we proposed a coverage metric measuring how well a test covers a LTL requirement. The vacuity-based coverage criterion requires a test suite to check the relevancy of each subformula of a LTL property to a system. Whalen and Rajan *et al* [26,20] described a similar strategy, presenting 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 the execution traces. Fraser and Gargantini conducted a comparison of these techniques in [7].

The key element of our approach is test criteria based on the coverage for Büchi automaton. Fujiwara *et al.* [8] proposed the partial W-method for test case selection. They evaluate the adequacy of a test suite with respect to the coverage for a finite state machine. Besides the obvious difference between Büchi automaton and finite state machine, the automaton in our approach is used to specify the requirement of a system, whereas the automaton used by Fujiwara *et al.* is the model of a system.

This study extends our previous work on specification-based testing with temporal logics [23]. In [23] we developed the coverage criteria for Linear Temporal logic (LTL) based the notion of (non-)vacuousness. One of its features and also a drawback is that the criteria depend heavily on the syntactical structures of a LTL formulae, which makes the approach susceptible to syntactical variance of formula, even though the formula may have the exactly same semantics. For instance, 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. Our automaton-based coverage criteria are proposed to overcome this problem of syntactical dependency. Büchi automaton captures the semantics of a linear-temporal requirement, and methods such as automaton minimazation may be used for removing the syntactical difference between automata of the same semantics. In [22] we also investigated state coverage criteria, and conducted a computational study for them [28]. In the paper we focus on the transition of a Büchi automaton. We show that that the new criteria out-perform the state-coverage criteria, both in theory and in our experiments. We also perform an extended computational study that now includes the sensitivity analysis using fault injection technique.

2 Preliminaries

2.1 Kripke Structures, Traces, and Tests

We use *Kripke structures* to model the systems. A Kripke structure is a finite transition system in which each state is labeled with a set of atomic propositions. Atomic propositions represent primitive properties held at a state semantically. Definition 1 defines Kripke structures formally.

Definition 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 $\mathcal{A}$, and 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.

The following notations are used to represent 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 (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 3 (Test and Test Suite). *A test, or a test case, 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.*

2.2 Generalized Büchi Automata

Definition 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 4 using state labeling approach in [9] with one modification: we label the state with a set of literals, instead of with a set of sets of atomic propositions in [9]. 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 5 for (labeled) generalized Büchi automata (GBA).

Definition 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 $\mathcal{L} : S \to 2^P$ 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 $K \models \phi$.

3 Transition Coverage Metrics and Criteria

Definition 6 (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 $s \to s'$ 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 $s \to s'$ 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 allows non-deterministic transitions, a test may induce more than one successful runs of B. We define a *weakly* covered transition as a transition appearing on some successful runs induced by t, and a *strongly* covered transition as a weakly covered transition appearing on every successful run induced by t. It shall be noted that by requiring t satisfying B, the strong coverage requires that t induces at least one successful run γ of B. Since a strongly covered transition $s \to s'$ appears on every successful run induced by t, it shall also appear on γ. Hence, by Definition 6 a strongly covered transition must also be a weakly covered one.

Definition 7 (Weak Transition Coverage 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 $\delta' = \{s \to s' \mid (s \to s') \in \delta \wedge \exists t \in T.(t \text{ weakly covers } (s \to s'))\}$. T weakly covers δ if and only if $\delta' = \delta$.*

Definition 8 (Strong Transition Coverage 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 $\delta' = \{s \to s' \mid (s \to s') \in \delta \wedge \exists t \in T.(t \text{ strongly covers } (s \to s'))\}$. T strongly covers δ if and only if $\delta' = \delta$.*

Theorem 1 shows that the strong transition coverage criterion subsumes the weak transition coverage criterion.

Theorem 1. *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 8, 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 8, T also weakly covers $\mathcal{S}$. □

It shall be noted that covering transitions of a GBA is very different from similar practice on a Finite State Machine (FSM). In this paper, we do use Kripke structure, which is essentially a FSM, to model a system. Meanwhile, the criteria defined above focus on covering a property in the form of Büchi automata. By shifting the focus towards the property, we are able to measure the semantic adherence of a system with respect to the property, instead of simply going through the structural elements of the system. By examining the GBA, we are able to reason and test subtle temporal behaviors of the system that cannot be captured by a FSM.

4 Model-Checking-Assisted Test Generation for Transition Coverage

To improve the efficiency of test case generation, we develop model-checking-assisted test case generation algorithms based on off-the-shelf Büchi-automaton-based model checkers. Model checking with Büchi automaton is a well-studied subject (cf. [9]). Efficient algorithms and tools, such as SPIN [13], have been developed over the years. In general a Büchiautomaton-based model checker verifies a system K against a property ϕ in two stages: first, a Büchi automaton $B_{\neg\phi}$ is constructed for the negated property $\neg\phi$; second, the checker generate the product of $B_{\neg\phi}$ and K, and then run an emptiness check on the product automaton. If the result is positive, 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, our test-case generation algorithms are not tied to any specific model checker. We designate a routine $MC\_isEmpty$ in the algorithms as an abstraction of the core emptiness checking routine of a Büchi-automaton-based model checker. $MC\_isEmpty(B, K)$ checks the emptiness of the product of a Büchi automaton B and a Kripke structure K. $\overline{B}$ accepts K if the product is empty; 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. Throughout the process, B, as an input to the $MC\_isEmpty(B, K)$, is a Bc̈uhi automaton that encodes a test objective for a test criterion, and the trace that returns from the emptiness check may be used to construct a test case for that test objective.

A central question in model-checking-assisted test case generation is how to encode test objectives as temporal properties acceptable to a model checker. In our case, these temporal properties, referred to as "trap" properties, characterize test cases covering the transitions of a Büchi automaton. We develop a method based on graphic transformation, to derive "trap" properties from a Büchi automaton under transition coverage criteria. 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 9). 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 10). Figure 1 shows the workflow of model-checking-assisted test case generation under both transition coverage criteria for Büchi automaton.

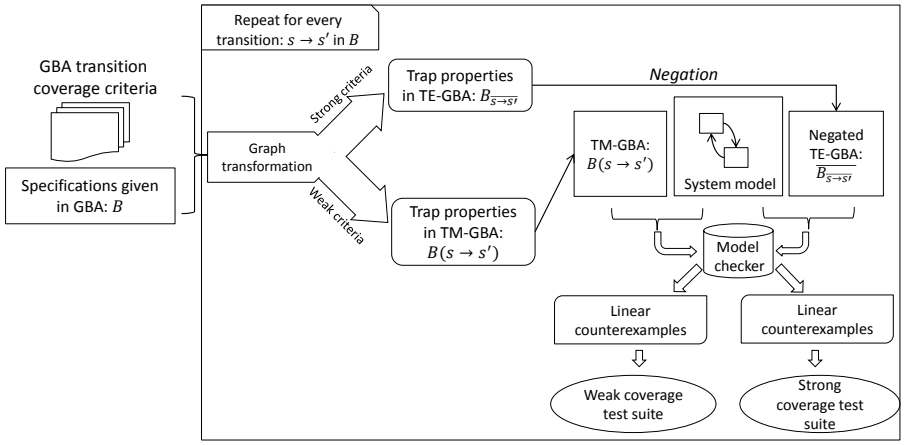

Fig. 1. The workflow of model-checking-assisted test case generation under transition coverage criteria for generalized Büchi automaton (GBA).

Definition 9 (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,*

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

For a Büchi automaton B and a transition $s \to s'$ of B, the corresponding TM-GBA $B(s \to s')$ contains two copies of the original B, with additional indices attached to the states of these copies with 0 and 1, respectively. The copies are linked by the transition $\langle s,0\rangle \to \langle s',1\rangle$. The start states of B remain in the first copy, while only the second copy has all the acceptance states. Therefore, a successful run of $B(s \to s')$ must travel from a start state in 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 $s \to s'$.

As an example, consider the LTL property $\mathbf{G}(\neg t \Rightarrow ((\neg p\ \mathbf{U}\ t) \vee \mathbf{G}\neg p))$. The property specifies a temporal requirement for the GIOP model, the general Inter-Object Request Brokers (ORB) Protocol [17], which we used in our computational experiments. In this formula, the atomic proposition t stands for a request being sent in the model, and p stands for an agent receiving a response. The formula states that an agent would not receive any response until a request has been made. Figure 2 is a generalized Büchi automaton for the LTL property. Figure 3 shows a TM-GBA covering the transition $s_0 \to s_1$ for the GBA in Figure 2.

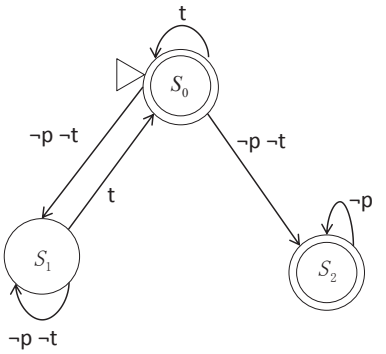

Fig. 2. A general Büchi automaton representing the LTL property $\mathbf{G}(\neg t \Rightarrow ((\neg p\ \mathbf{U}\ t) \vee \mathbf{G}\neg p))$.

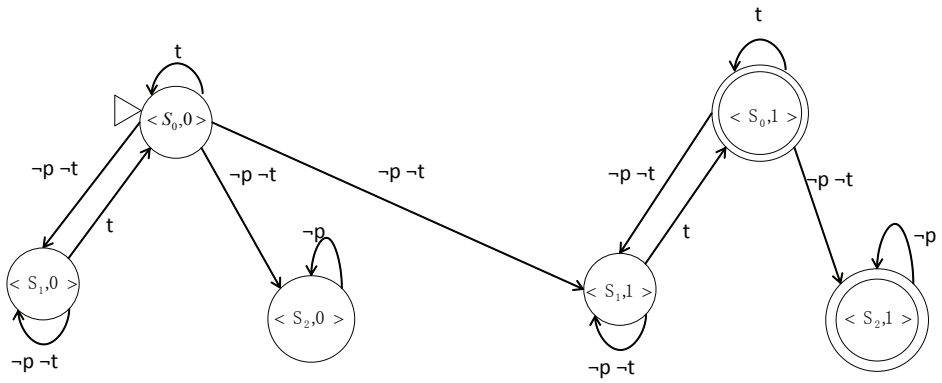

Fig. 3. A transition marking general Büchi automaton covering the transition $s_0 \to s_1$ for the GBA in Figure 2.

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

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;
1: **for** every $s \to s' \in \Delta$ **do**
2: Construct a TM-GBA $B(s \to s')$ from B that marks the transition $s \to s'$;
3: $\tau = MC\_isEmpty(B(s \to s'), K_m)$;
4: **if** $|\tau| \neq 0$ **then**
5: $ts = ts \cup \{\mathcal{V}(\tau)\}$
6: **else**
7: **return** $\emptyset$;
8: **end if**
9: **end for**
10: **return** ts;

Algorithm 1 generates a test suite that weakly covers all transitions of a GBA B. For every transition $s \to s'$ of B, the algorithm builds a corresponding TM-GBA $B(s \to s')$. The algorithm then invokes $MC\_isEmpty$ on $B(s \to s')$ and a system model K_m, which will check the emptiness of the product of the inputs. If the product is not empty, $MC\_isEmpty$ returns a successful run of the product of $B(s \to s')$ and the system model K_m. A test case related to this successful run is then added to the test suite. Theorem 2 shows the correctness of Algorithm 1.

Theorem 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 $s \to s'$ and $MC\_isEmpty$ $(B(s \to s'), K_m)$ returns a successful run of the production of $B(s \to s')$ and

K_m such that $\mathcal{V}(\tau) = t$. Since any successful run of the production of $B(s \to s')$ 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 $s \to s'$ and a successful run τ of the production of $B(s \to s')$ and K_m such that $\mathcal{V}(\tau) = t$. We will show that t weakly covers $s \to s'$. By Definition 7, we need to show that there is a successful τ' of B such that τ' takes the transition $s \to s'$. 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 $B(s \to s')$ 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 $B(s \to s')$, 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 9, each transition in $B(s \to s')$ is mapped to a transition in B and each acceptance state in $B(s \to s')$ is mapped to an acceptance state in B. In addition, τ has to go through $\langle s, 0\rangle \to \langle s', 1\rangle$ because, by Definition 9, 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 $B(s \to s')$ is successful is that it has to go through $\langle s, 0\rangle \to \langle s', 1\rangle$. Therefore, τ' has to take $s \to s'$, and we proved (ii). □

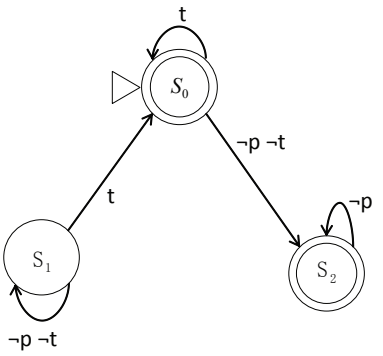

Fig. 4. A transition excluding general Büchi automaton covering the transition $s_0 \to s_1$ for the GBA in Figure 2.

For strong transition coverage criterion, a trap property is a transition excluding generalized Büchi automaton (TE-GBA). For a Büchi automaton B and a transition $s \to s'$ of B, the TE-GBA $B_{\overline{s \to s'}}$ is constructed 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.

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

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 all the transitions of B and K_m passes ts. Return $\emptyset$ if such a test suite is not found;
1: **for** every $(s \rightarrow s') \in \Delta$ **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: **return** $\emptyset$;
8: **end if**
9: **end for**
10: **return** ts;

Definition 10 (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 $B_{\overline{s \rightarrow s'}} = \langle P, S, S_0, \Delta - \{s \rightarrow s'\}, \mathcal{L}, \mathcal{F}\rangle$.*

Algorithm 2 generates a test suite that strongly covers the transitions of a given GBA B. Similarly to the weak coverage, the algorithm builds a TE-GBAfor every transition of B. 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 $\overline{B_{\overline{s \rightarrow s'}}}$ and the system model K_m is not empty, the function *MC_isEmpty* returns a successful run of the product of $\overline{B_{\overline{s \rightarrow s'}}}$ and the system model K_m. Since this run is also a trace of K_m, it is then mapped to a test case consisting of a series of atomic propositions, and added to the resulting test suite. Theorem 3 shows the correctness of Algorithm 2.

Theorem 3. *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 $s \rightarrow s'$ and *MC_isEmpty* $((\overline{B_{\overline{s \rightarrow s'}}}), K_m)$ returns a successful run of the product of $\overline{B_{\overline{s \rightarrow s'}}}$ and K_m such that $\mathcal{V}(\tau) = t$. Since any successful run of the production of $\overline{B_{\overline{s \rightarrow s'}}}$ 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 $s \rightarrow s'$ and a successful run τ of the production of $\overline{B_{\overline{s \rightarrow s'}}}$ and K_m such that $\mathcal{V}(\tau) = t$. We will show that t strongly covers the transition $s \rightarrow s'$.

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 $s \rightarrow s'$ 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 $B_{\overline{s \to s'}}$. 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. □

Complexity Analysis 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. [25]). 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 [19] 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 6: 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 $s \to s'$.

5 Transition-Coverage-Induced Requirement Refinement

Our coverage criteria and metrics not only measure the quality of a test suite in terms of its relevancy to a requirement in Büchi automaton, they may also be used to identify the deficiency of the requirement itself. Insufficient coverage may be caused by a problem in a system design, but it may also be due to the deficiency of the requirement itself. For example, the requirement may be imprecise and/or too general with respect to the system design. The information from our model-checking-assisted test case generation algorithm may be used to refine the requirement.

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 3, 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

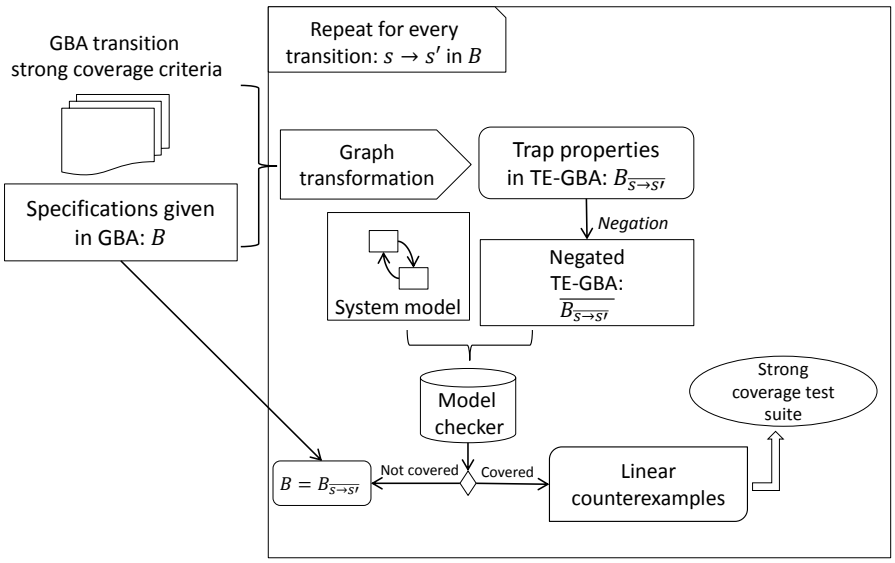

Fig. 5. The workflow of strong transition coverage criterion based GBA refinement.

Büchi automaton. The resulting property B' will be more semantically restricted than the original property, that is, $B' \sqsubseteq B$.

Lemma 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 10 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 4. *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 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 11 (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 11 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 1. *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 11, $s \to s'$ 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 6, 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 1 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, in other words, 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

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** $\langle B, ts\rangle$;

that $K \models B$ and $K \not\models B_{\overline{s \rightarrow s'}}$, but testing alone may not be able to conclusively prove that K satisfies B. Nevertheless, lack of the strong coverage for transition $s \rightarrow s'$ indicates that $s \rightarrow s'$ is a vacuous transition for K. Therefore, we can remove $s \rightarrow s'$ 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 that differs on line 7. Instead of returning with a failed attempt in Algorithm 2 when the full strong state coverage cannot be achieved, Algorithm 3 refines the input GBA by removing vacuous transitions immediately. The output will be a GBA refined and without vacuous transitions with respect to the model, as well as a test suite for the refined GBA. Figure 5 depicts the workflow of the GBA refinement process.

6 Experiment

6.1 Experiment Settings

To obtain a close-to-reality measurement of the performance of our coverage criteria, we purposely select the subjects of our experiments from three different fields. The first subject used in our experiment 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 [17]. 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 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 [16] from a classic Simulink [21] fuel system demo model.

Each model has a set of linear temporal properties that specify behavior requirement for the underlying system. For the GIOP model, the property models the requirement on the behaviors of the recipient agent during communication. The LTL property for the Needham-Shroeder public-key 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 fuel system checks that under abnormal conditions, the system's fault tolerant mechanism is able to function properly without a disastrous failure.

Table 1 provides an overview of the models and properties, showing the size of both the models and properties in terms of the number of branches, states/transitions of the LTL property equivalent Büchi automata, and atomic propositions in the properties. All of the information in Table 1 are of relevance to the diversified profiles of test criteria we used in the experiments for the comparison, in terms of the size of test suites generated.

Table 1. Overview of the models and properties used in the experiments.

Models	Branches	States	Transitions	Atoms.
GIOP	70	2	6	4
Needham	43	2	6	3
Fuel	55	4	21	4

We select several traditional as well as specification-based testing criteria as basis for performance comparison. Based on the coverage for outcomes of a logic expression (c.f. [15]), *Branch coverage* (BC) is one of the most commonly-used structural test criteria. We include a strong state coverage criterion (SC/strong) and a weak state coverage criterion (SC/weak) for Büchi automaton[22]. We also include a property-coverage criterion (PC) for Linear Temporal Logic (LTL) [23]. In our experiment, the performance of these criteria, and two transition coverage criteria (TC/stong and TC/weak) are compared with each other.

6.2 Methodologies

To assess the effectiveness of our transition-coverage criteria, we carry out an extended computational study using two different methodologies: cross-coverage analysis and fault-injection-based sensitivity analysis.

Cross-coverage analysis the cross-coverage measures how well a test suite generated for a test criterion covers another test criterion. The cross coverage is used as an indicator for the semantics strength of a test criterion with respect to others. In [28] 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 [24] to perform graph transformation required for building TE-GBAs and TM-GBAs, and adopt SPIN [13] as the underlying model checker to assist test case generation. For the other criteria, we generate trap properties in LTL for BC/PC, and state coverage criteria based trap GBAs for SC/strong and SC/weak. Interested readers can refer to [28,29] for the details of the tool that we developed for cross-coverage comparison between test criteria.

Fault-injection-based sensitivity analysis Fault-inject technique (c.f. [27]) is a classic techniques used in software engineering for evaluating the sensitiveness of a quality assurance tool towards injected faults. The sensitivity analysis checks the effectiveness of a test suite under a given criterion. The effectiveness is measured by the test suite's ability of catching faults systematically introduced to a system. A higher number of faults being caught indicates that the underlying test criterion is more sensitive in catching faults. To inject faults to a model, we mutate relational operators (e.g. changing $\geq$ to $<$), one operator at a time. We then run on a faulty model a test suite generated for a test criterion. If the execution of the faulty model under a test case is different from that of the original model, then the injected fault is caught by the test case, that is, the underlying test criterion is sensitive enough to catch the fault.

6.3 Experiment Results and Analysis

Both cross coverage analysis and sensitivity analysis require the test suites generated for test criteria being involved. Table 2 shows the measurement of the generated test cases. For the branch coverage (BC), we first generate tests specifically for every single branch in the model. Based on the coverage information, we select two more groups of test cases. By using an Integer Linear Programming solver, we obtain an optimal test suite that consists of the least number of test cases and covers the maximum number of branches that can be covered (BC(Opt.)). This test suite represents the theoretical lower bound of the number of test cases needed for covering the system model under BC. The other test suite (BC(Grd.)) is selected using a greedy algorithm. For instance, if the first test case covers branches No. 2 and 3, then test cases for the second and third branch will no longer be generated, and so on. The greedy algorithm represents the common practice of selecting a near-optimal test suite, to reduce the cost of test execution. "TS Size" in Table 2 indicates the number of test cases each test suite has, and "Max./Min./Avg. Length" specifies the length of the lasso-shaped test cases, i.e., the number of steps in the counterexample trace produced by the model checker. Finally, "Gen. Time" and "Exec. Time" represent the time it took to generate the traces and execute the test cases, respectively.

It shall be noted that for practical purpose, we enforce a time limit for the model checking process. This is due to the fact that SPIN suffers from "state space explosion problem" as an explicit state model checker [10]. SPIN may

run out of resources (time and/or space) before reaching a conclusive result. . Subsequently, we expect three possible outcomes of the model checking process: 1) returning with a counterexample trace, 2) returning with a answer that there is no counterexample or 3) terminating without returning value. For the third case, we count the time limit towards the generation time, which explains why some entries in Table 2 is significantly longer than the other criteria.

Table 2. Test suites overview

Model	GIOP							
	BC	BC(Opt.)	BC(Grd.)	PC	SC/strong	SC/weak	TC/strong	TC/weak
TS Size	54	4	10	3	2	2	6	6
Max. Length	779	779	779	601	602	602	602	602
Min. Length	34	605	49	280	602	572	602	572
Avg. Length	405	664	534	494	602	587	602	588
Gen. Time	1087	27	51	332	0.02	3.7	0.06	13
Exec. Time	0.586	0.05	0.1	0.06	0.03	0.05	0.11	0.13
Model	Needham Protocol							
	BC	BC(Opt.)	BC(Grd.)	PC	SC/strong	SC/weak	TC/strong	TC/weak
TS Size	37	7	13	3	2	2	6	6
Max. Length	70	70	62	34	43	42	43	42
Min. Length	9	22	22	33	41	41	41	41
Avg. Length	43	51	48	34	42	42	42	41
Gen. Time	360	0.065	0.122	0.03	0.02	0.02	0.06	0.06
Exec. Time	0.335	0.063	0.118	0.03	0.02	0.02	0.06	0.06
Model	Fuel System							
	BC	BC(Opt.)	BC(Grd.)	PC	SC/strong	SC/weak	TC/strong	TC/weak
TS Size	45	1	8	3	2	2	6	7
Max. Length	52904	52904	9261	8594	8482	538	8482	5320
Min. Length	27	52904	29	130	1530	254	174	192
Avg. Length	3985	52904	1975	4239	5006	396	4661	2003
Gen. Time	602	0.76	0.155	0.178	600	600	780	720
Exec. Time	751	150	0.375	0.379	0.24	0.02	0.61	0.3

Table 3 shows the results from the cross-coverage analysis. The number in each cell indicates the coverage of test cases generated for the criterion on the row w.r.t. the criterion on the column. That is, we first generate the test suites for the criterion on a row, and then measure the coverage of the test suite on the test criterion on a column. Numbers on diagonal cells are marked with parentheses. They represent the coverage of a test suite generated for the same criterion. A less-than perfect coverage on these diagonal cells may be caused by any of

the following reasons: 1) it indicates potential deficiency of a model and/or a requirement, or 2) the model checker could not terminate within the time limit. For instance, for the fuel system model, the test suite for the strong transition coverage criterion may only reach 29% coverage upon all the transitions. This is due to both the existence of vacuous transitions in the specification, as well as the model checking process not returning with an conclusive answer, i.e. the second and third outcome of the process as explained above.

Table 3. Cross-coverage comparison results

	GIOP					
	BC	PC	SC/strong	SC/weak	TC/strong	TC/weak
BC	(77%)	75%	100%	100%	100%	100%
PC	66%	(75%)	100%	100%	100%	100%
SC/strong	66%	75%	(100%)	100%	100%	100%
SC/weak	66%	75%	100%	(100%)	100%	100%
TC/strong	66%	75%	100%	100%	(100%)	100%
TC/weak	66%	75%	100%	100%	100%	(100%)
	Needham Protocol					
	BC	PC	SC/strong	SC/weak	TC/strong	TC/weak
BC	(86%)	100%	100%	100%	100%	100%
PC	47%	(100%)	100%	100%	100%	100%
SC/strong	47%	100%	(100%)	100%	100%	100%
SC/weak	28%	0%	0%	(100%)	0%	100%
TC/strong	47%	100%	100%	100%	(100%)	100%
TC/weak	40%	0%	0%	100%	0%	(100%)
	Fuel System					
	BC	PC	SC/strong	SC/weak	TC/strong	TC/weak
BC	(82%)	25%	75%	50%	86%	33%
PC	78%	(100%)	50%	50%	29%	33%
SC/strong	75%	100%	(50%)	50%	29%	33%
SC/weak	64%	25%	75%	(50%)	86%	33%
TC/strong	75%	100%	100%	50%	(29%)	33%
TC/weak	67%	25%	75%	50%	86%	(33%)

The proposed transition coverage criteria, especially, its strong variance, place second in the cross coverage analysis, only after the branch coverage criterion. It shall be noted that the branch coverage criterion is a white-box testing criterion. The criterion forces test cases for every branch of a system, and hence the test suite for the branch coverage criterion is in general larger than those for transition coverage criteria, as evident in 2. A smaller test suite, coupled with a good performance in cross-coverage analysis, could make the transition coverage

criteria a competitive alternative to a white-box coverage criterion such as the branch coverage criterion.

It is also noted that the test suites for transition coverage criteria do not achieve the full branch coverage. The is mainly due to the fact that we only use one temporal property for each model, which do not fully cover all the functional aspects of the models. For instance, the property for the GIOP model specifies the requirement for the recipient's behavior when it is waiting for or receives a message, and does not concern other functions. Therefore, the test cases bypass some code segments and lead to a less-than perfect branch coverage.

This leads to another important observation for 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. We capitalized this observation via our transition-coverage-induced property refinement in Section 5. 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 also establish that transition coverage criteria out-perform the other property-based coverage criteria, albeit slightly. From a superficial point of view, there are usually more transitions than states in a GBA, or the atomic propositions in the equivalent LTL formula. Thus, a transition coverage based test suite can potentially address more scenarios. We can also see a decent correlation between the state and transition coverage based test suites, for both the strong and weak variants. This proves that we are able to further remove the syntax dependency that still exists for the property coverage criterion in [23], therefore come one step closer to the semantic essence of the temporal properties. It is also straightforward that a transition coverage criterion subsumes its state coverage counterpart, since covering a transition means covering both the source and the target states. Last but not least, with the fact that transition coverage criteria are tailored towards the subtle differences among transitions, the refinement process is able to yield a finer tuned refined GBA that the state coverage criteria are unable to produce.

Table 4 shows the results of fault-injection-based sensitivity analysis. Faults are introduced by relational operators mutation. The total count of "faults", i.e., the number of relational operators available in the models that can be mutated, are specified in the parenthesis along side the name of the model. We then list out the percentage of the faults that can be detected by the test suites we generated towards the different test coverage criteria. We also define a *Sensitivity Adjusted Cost* (SAC) as follows,

$$SAC = \frac{\text{(Total Length of the Test Suite)}}{\text{(Percentage of Detected Faults)}}$$

Table 4. Injected faults detection results

Total Faults	GIOP (49)		Needham Protocol (24)		Fuel System (191)	
	Detection Rate	SAC	Detection Rate	SAC	Detection Rate	SAC
BC	76%	28776	92%	1729	66%	118355
BC(Opt.)	76%	3495	79%	452	N/A	N/A
BC(Grd.)	76%	7026	88%	709	66%	23939
PC	67%	2212	75%	136	76%	16733
SC/strong	67%	1797	83%	101	69%	14510
SC/weak	67%	1752	25%	336	54%	1467
TC/strong	67%	5391	83%	304	69%	40530
TC/weak	67%	5266	38%	647	58%	24174

Note that the cost of executing a test suite is in general proportional to the size of the test suite. The SAC essentially means the adjusted cost of test (execution) w.r.t. the sensitivity of the underlying test criterion.

In all three models, the property-based criteria, including transition coverage criteria, are able to detect a good portion of the injected faults. Because of the code-based nature of branch coverage, it is to be expected that BC would have the best detection rate. However, in the case of the fuel system model, some of the test cases are too long to be executable (with the largest length over 50000, see Table 2). As a result, both the full and greedy test suites can only detect two thirds of the faults, while other criteria based test suites catch up, or even surpass it with fewer and shorter test cases. Judging by SAC, we can see that PC and SC tend to have lower cost, while TC would be slightly more expensive due to there being more test cases.

Also, in the case of the Needham-Schroeder protocol, we can see that the strong variants of both SC and TC out-perform the weak counterparts rather significantly, which can be explained by the less demanding nature of weak criteria producing shorter test cases, thus are less effective in finding the faults. Furthermore, once again we can see the correlation between SC and TC, as well as TC/weak slightly out-performs SC/weak, which is comparable to our previous analysis on the cross-coverage experiments.

7 Conclusions

We considered specification-based testing for linear temporal properties expressed in generalized Büchi automata (GBAs). We introduced two variants of test metrics and criteria based on the test-case coverage for transitions of a Büchi automaton. The test metrics measure the relevancy of test cases with respect to a Büchi automaton, and the test criteria may be used for selecting test cases for testing a system with respect to the Büchi automaton. We developed a model-checking-assisted algorithm to automate test case generation for proposed criteria.

Although specification-based testing with automata has been studied before (c.f.[8]), the subject of these previous works is a system design modeled in finite automaton. In comparison, we focus on high-level requirement encoded in Büchi automaton. While these previous works studied testing techniques that checked the conformance of an implementation to a system design, we explored technique that examined the conformance of a system design with respect to its requirements. Moreover, with the power of Büchi automata, we are able to study specification-based testing for linear temporal properties that model complex and infinite linear-time behaviors of an reactive system.

The transition-coverage criteria not only measure the quality of a test suite in terms of its relevancy to a requirement in Büchi automaton, they may also be used to identify the deficiency of the requirement itself. We defined the notion of vacuous transitions. By removing these vacuous transitions, we obtain a refined requirement that more closely describes the behaviors of a system. Using the feedback from model-checking-assisted test case generation, we developed an algorithm to identify vacuous transitions and to refine the requirement.

To assess the effectiveness of the proposed approach, we carried out an extended computational study using two different methodologies: the cross coverage analysis measures how well a test suite generated for a test criterion covers another test criterion. The cross coverage is used as an indicator for the semantic strength of a test criterion with respect to others; the sensitivity analysis checks the effectiveness of a test suite for a given criterion, in terms of the ability of the test suite catching faults systematically introduced to a system. In both cases, our transition-coverage-based criteria show a better performance over existing test criteria. A particular strength of our criteria is that the size of a test suite selected by our criteria is relatively small. The small size of these test cases not only reduces the cost of test execution, it also helps testing activity centeralized on system requirement.

References

1. Bringmann, E.; Krämer, A.: Model-based testing of automotive systems. In: 2008 International Conference on Software Testing, Verification, and Validation (ICST. pp. 485–493 (2008)
2. Calvagna, A., Gargantini, A.: A Logic-Based Approach to Combinatorial Testing with Constraints. In: 2nd International conference on Tests and Proofs (2008)
3. Chilenski, J.J., Miller, S.P.: Applicability of modified condition/decision coverage to software testing. Software Engineering Journal 9(5), 193–200 (1994)
4. Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic. In: Kozen, D. (ed.) Proceedings of the Workshop on Logic of Programs, Yorktown Heights. Lecture Notes in Computer Science, vol. 131, pp. 52–71. Springer-Verlag (1981)
5. Committee, S..: Software Considerations in Airborne Systems and Equipment Certification. Tech. rep., Radio Technical Commission for Aeronautics (1992)
6. Dahl, O.J., Dijkstra, E.W., Hoare, C.: Structured Programming, A.P.I.C. Studies in Data Processing, vol. 8. Academic Press (1972)

7. Fraser, G., Gargantini, A.: 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. IEEE Computer Society, Washington, DC, USA (2009)
8. Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Trans. Softw. Eng. 17(6) (Jun 1991)
9. Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: PSTV'95. pp. 3–18. Chapman and Hall (1995)
10. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: In Protocol Specification Testing and Verification. Chapman & Hall (1995)
11. Giannakopoulou, D., Lerda, F.: From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata. In: Lecture Notes In Computer Science; Vol. 2529. Springer-Verlag (2002)
12. Heimdahl, M., Rayadurgam, S., Visser, W.: Specification centered testing. In: Proceedings of the Second International Workshop on Automated Program Analysis, Testing and Verification (2001)
13. Holzmann, G.J.: The Model Checker SPIN. IEEE Transactions on Software Engineering 23(5), 279–295 (May 1997)
14. Hong, H.S., Lee, I., Sokolsky, O., Ural, H.: A temporal logic based theory of test coverage and generation. In: TACAS'02. LNCS, vol. 2280, pp. 327–341 (2002)
15. Jorgensen, P.C.: Software Testing: A Craftsman's Approach. CRC Press, Inc., Boca Raton, FL, USA, 1st edn. (1995)
16. Joseph, S.: Fault-Injection through Model Checking via Naive Assumptions about State Machine Synchrony Semantics. Master's thesis, West Virginia University, Morgantown, West Virginia (1998)
17. Kamel, M., Leue, S.: Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and SPIN. International Journal on Software Tools for Technology Transfer (STTT) 2(4) (Mar 2000)
18. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. International Journal on Software Tools for Technology Transfer (STTT) 4(2) (Feb 2003)
19. Michel, M.: Complementation is more difficult with automata on infinite words. CNET, Paris, France (1988)
20. Rajan, A.: Coverage metrics for requirements-based testing. Ph.D. thesis, University of Minnesota, Minneapolis, MN, USA (2009)
21. SIMULINK: Dynamic system simulation for matlab, the mathworks (January 1997)
22. Tan, L.: State Coverage Metrics for Property Coverage Testing with Büchi Automata. In: 5th International Conference on Tests and Proofs. Lecture Notes in Computer Science, Springer Verlag, Zurich, Switzerland (2011)
23. Tan, L., Sokolsky, O., Lee, I.: Specification-based Testing with Linear Temporal Logic. In: IRI'04. pp. 493–498. IEEE society (2004)
24. Tsay, Y.k., Chen, Y.f., Tsai, M.h., Wu, K.n., Chan, W.c.: 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)
25. Vardi, M.: Automata-theoretic model checking revisited. In: 8th Verification, Model Checking, and Abstract Interpretation. Springer (2007)

26. Whalen, M.W., Rajan, A., Heimdahl, M.P., Miller, S.P.: Coverage metrics for requirements-based testing. In: Proceedings of the 2006 international symposium on Software testing and analysis. ISSTA '06, ACM, New York, NY, USA (2006)
27. Young, M., Pezze, M.: Software Testing and Analysis: Process, Principles and Techniques. John Wiley & Sons (2005)
28. Zeng, B., Tan, L.: Test Criteria for Model-Checking-Assisted Test Case Generation: A Computational Study. In: International Conference on Information Reuse and Integration. IEEE (2012)
29. Zeng, B., Tan, L.: A unified framework for evaluating test criteria in model-checking-assisted test case generation. Information Systems Frontiers pp. 1–12 (April 2013)