http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (14): 2943-2949, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.2943.2949
© 2013 Asian Network for Scientific Information

A Novel Automatic Testing System Close-loop Model Based on Event Feedback

[1,2]Shuai Wang, [2]Yindong Ji, [2]Wei Dong, [2]Xinya Sun and [3]Yafang Liu
[1]Department of Computer Science and Technology, Tsinghua University
[2]Tsinghua National Laboratory for Information Science and Technology, Tsinghua University
[3]China Academy of Space Technology

Abstract: Tsting is essential for system verification and quality assurance. Most of the obtained results on testing focus on how to generate test cases and how to achieve a given goal optimally with a testing strategy. However, how to carry out testing automatically may be another very important and useful problem. In this field, how to model the automatic testing system and how to design the automatic test controller based on the model are the two key problems which are similar with the core problems of automatic control theory. Now the research reports on this direction are very few. In this study, we proposed a novel automatic testing system model based on controlled automata. The test controller, system under test and test observer compose a close-loop testing system based on event feedback. Meanwhile the construction methods of the automatic test controllers were given. We also captured the control mechanism in the algebraic framework of parallel composition of automata for the close-loop testing system.

Key words: Automatic testing system model, automatic test controller, close-loop testing system, cotrolled automata

INTRODUCTION

Testing is an important work to provide information about the quality and performance of system under test, with respect to certain context. Testing, for different test objective may be implemented at different time in the system development process with different method. A lot approaches have been proposed for system testing. But, testing is still the least understood part of the system development process (Whittaker, 2000). Until now, testing quality is highly depends on experience of tester.

Most of the obtained results focus only on the following questions: given a testing technique or test data adequacy criterion, how to generate test case and/or what properties (defect detection ability, test data complexity and cost) the approach may have (Poore, 2000; Gutjahr, 1999; Chen and Yu, 1996). At the same time, another important question also has gained some attentions (Tal *et al*., 2002; Sahinoglu, 2003; Chen and Yu, 2002; Cai, 2002): Given a test goal, how to achieve this goal optimally with a testing strategy. A test strategy determines how to select test cases to be applied to the system under test and decide the stopping time of testing process to achieve test goal.

Because system under test is more and more complex, automatic testing become a very important approach for system verification. The automatic testing method can increase test efficiency and decrease test cost. With this point of view, several automatic testing systems for different systems under test are designed and implemented (Gao *et al*., 2010; Meyer *et al*., 2007; Liu and Nakajima, 2011). These studies have shown the fact that automatic testing can provide important benefits for system testing practice. But most of existing methods only consider the implementation of automatic testing for certain system. There is no report on the basic theory framework of automatic testing, such as the mathematical model of automatic test system and the design method of automatic test controller.

In this study, we focus on the basic theory framework of automatic testing for the man-made system which can be modeled as an automaton. Considering the lack of automatic testing system model, we proposed a novel automatic testing system model based on controlled automata. In this model, the test controller, system under test and test observer compose a close-loop system based on event feedback. Meanwhile, we gave the mathematical description of the close-loop system. Based on the model, we also gave a construction method of automatic test controller.

The rest of this study is organized as follows. The traditional automata-based test method is introduced in section 2. In section 3 the theory framework of automatic testing based on event feedback is introduced. The

Corresponding Author: Shuai Wang, Department of Computer Science and Technology, Tsinghua University

example of our automatic testing framework on Automatic Train Protection system is given in section 4. Finally, the conclusion is presented in section 5.

TESTING WITH AUTOMATA

Man-made system usually cannot be described by differential or difference equations but can be modeled as Discrete Event System (DES) and then the behavior of system can be modeled as an automaton. The testing of this type system can be taken as checking the outputs when certain input sequence is applied. After the outputs observing at certain output ports, they will be compared with the expected outputs corresponding with the inputs. In the testing process of system, different test cases are selected to be applied to the system under test in accordance with a given testing strategy to fulfill a test criterion.

For the sake of convenience, we will recall the basic idea of automata-based system testing approach in the follows. Let us first see the definition of automaton that is used to model the system under test as below.

Definition 1: A mealy automaton is a seven-tuple $G = (Q, q_0, Q_m, \Sigma, \Lambda, \delta, \lambda)$ (Hopscroft *et al.*, 2000; Lee and Yannakakis, 1996):

- **Q:** Is the set of states
- **Σ:** Is the finite set of input alphabet and one input alphabet describes one input event
- **Λ:** Contains all output events and one output event is described by one output alphabet;
- **δ: Q×Σ→Q:** Is the state transition function
- **q_0:** Is the initial state
- **$Q_m \subset Q$:** Is the set of finite marked states and the marked state is usually used to describe the state when one task finishing or one special event executing
- **λ: Q×Σ→Λ:** Is the output function

The verification of system is implemented through checking the output and the tail sate of every transition. The process for testing a specified transition from state "q_i" to state "q_j" with input/output " i_k/o_l" takes place in the following three steps:

- Implementation according to system specification is leaded into state "q_i"
- Input "i_k" is applied and the output is observed and checked, to see whether it is the expected output "o_l", or not.
- New state of implementation is checked to verify that if the tail state of the specified transition is "q_j" as expected, or not

We assume that there exists a reset action which is applied to make the system return to its initial state. This assumption ensures that each test case is applied in the same state of the system implementation. The reset action might be some input sequences, or a single action such as reset action. We define the Unique Input/Output (UIO) sequence (Sabnani and Dahbura, 1988) which denotes the state uniquely as a status message. The tail state is verified by checking this message after one test case is applied.

In this study, the test case for transition $(q_i, q_j; i_k/o_l)$, is constructed based on the U-method (Sabnani and Dahbura, 1988) as follows:

- Constructing the reset action r for implementation I, so that I can return to its initial state when each test case is applied
- Generating the shortest transition sequence that can lead the implementation from the initial state to the state "q_i", namely preamble sequence "ts_{pre}"
- Applying the input "i_k" which can enable the transition to be tested
- Generating the status message for tail state "q_j", namely postamble sequence "ts_{post}"

For the implementation of system with the status message feature, the test case for each transition is of the form:

$$r; ts_{pre}; t; ts_{post} \quad (1)$$

where, r is the reset action, ts_{pre} is the preamble sequence for the transition t, t: $q_j = \delta(q_i, a_k)$ is the transition to be tested, ts_{post} is the postamble sequence to check the tail state of transition t.

FRAMEWORK OF AUTOMATIC TESTING

System modelling with controlled automata: Classical automata G can describe the man-made system behaviors by the trace of event but when testing, we usually want to control the system performing at some intend manner to implement some test purpose. Especially some pre-defined event sequences will be checked for system testing. Then we adjoin a means of test control to the automata G, formally, defined as G_c. Then, G_c will play the similar role of controlled object when testing, which is another automaton, constructed from G by test specification.

Particularly, we construct the controlled Mealy automata based on the controlled automata proposed by Wonham (Ramadge and Wonham, 1987).

Definition 2: The controlled automaton for testing defined by an eight-tuple:

$$G = (Q, q_0, Q_m, \Gamma\times\Sigma, \Lambda, \delta_c, \lambda_c)$$

Where:

- Q, q_0, Λ have the same definitions with G
- Σ can be partioned into two disjoint subsets, the controllable event set Σ_c and the uncontrollable event set Σ_{uc}, $\Sigma = \Sigma_c \cup \Sigma_{uc}$ and $\Sigma \cap \Sigma_{uc} = \varnothing$. The controlled events are the events that can be enabled, or disabled by test controller. The uncontrolled events are the ones that cannot be prevented or disabled. We define $\Gamma = \{0, 1\}^{\Sigma c}$ as the binary assignment set for Σ_c. Each assignment is a control pattern. For $\gamma\in\Gamma$.

$$\gamma: \Sigma_c \rightarrow \{0, 1\} \quad (2)$$

1 denotes inputting test stimuli and 0 is opposite. This is different from the definition of Wonham:

- δ_c: $\Gamma\times\Sigma\times Q\rightarrow Q$ is the state transition function. For $\forall\sigma\in\Sigma$, $q\in Q$, $\gamma\in\Gamma$, the state transition function defined on the control patterns is:

$$\delta_c(\gamma, \sigma, q) = \begin{cases} \delta(\sigma, q) & \text{if } \delta(\sigma, q)! \text{ and } \gamma(\sigma) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3)$$

- $\delta(\sigma, q)$ stands for the function $\delta(\sigma, q)$ having a definition
- $\lambda: \Sigma x Q\rightarrow\Lambda$ is the output function. The output event is uncontrollable, such that we can say the output event will be certain to happen after the input is imposed. If the output is expected, the implementation is right

Note:

- $\gamma(\sigma) = 1$ means that event σ is enabled and $\gamma(\sigma) = 0$ means disabled
- In some conditions, an event would be modeled as uncontrollable, such as a change of sensor readings, hardware limitations and so on

Automatic testing model baded on event feedback: The formulation of the automatic testing considered in this study proceeds as follows. Consider the system under test modeled by a controlled Mealy automaton:

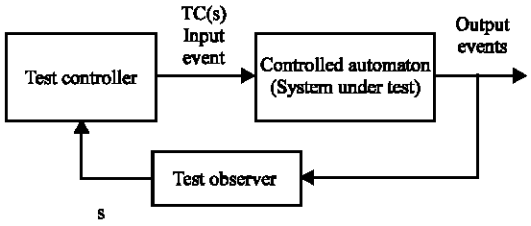

Fig. 1: Close-loop testing system based on event feedback

$$G_c = (Q, q_0, Q_m, \Gamma\times\Sigma, \Lambda, \delta_c, \lambda_c)$$

A test case usually defines a test logical which is composed of the sequence of inputs. The occasions when inputs are applied and the sequence of expected outputs are defined on the states of system or the events observed. This means that the implementation of automatic testing should require the feedback either the states or the events from the system under test. We adjoin a test controller, denoted by TC, to interact with G_c in an event feedback manner as depicted in Fig. 1. We assume that all the events in Λ are observed by test controller TC. Thus in Fig. 1 s is the string of all events outputted so far by G_c and is entirely seen by TC. Testing under partial event observation will not be discussed in this study.

The control paradigm is as follows. The behaviors of G_c can be controlled by TC by means of the controllable events in Σ_c. TC can enable or disable these events by defining different control patterns γ. Thus, a test controller TC can be seen as a machine that can map s to the power set of Σ_c.

$$TC: s\rightarrow 2^{\Sigma c} \quad (4)$$

where s is an event string outputted by G_c.

If $s\in L(G_c)$ is the event string outputted so far by G_c (under certain test case), then

$$TC(s)\cap H(q_0, s) \quad (5)$$

Is the active event set of G_c. $H(q_0, s)$ is a function to compute the active event set after execution of string s. TC (s) will prevent the event $\gamma(\sigma) = 0$ executing, even if it is in current active set.

In order to build up the close-loop testing system, we need to identify three components: the controlled object, a set of control policies and the switching rule of these policies. In this study, system under test is modeled as a controlled Mealy automaton. The test controller contains several test cases which are defined as separated control policies of the system and the defined execution strategy of these test cases.

Test controller design: The build of the test controller to perform a test task is based on the logical behavior of system and test purpose. As discussed in last section, the behavior of G_c can be controlled by TC in the sense that the controllable events. TC then can be abstractly defined as:

Definition 3: The test controller TC for the system G_c is a mapping from event sequence $s\in L(G_c)$ to the active input event set after string s, TC: $s\rightarrow 2^{\Sigma_c}$. Let us assume that formal language $K\subset L(G_c)$ is controllable, if the expected behavior of system resulted by certain test case set is $L(TC/G_c) = \overline{K}$, then TC is defined by:

$$TC(s) = \{\sigma\in\Sigma_c: (s\sigma\in\overline{K})\wedge(\sigma\subset H(q_0, s)\}$$

Moreover we eliminate two situations $\overline{K} = L(G_c)$ and $\overline{K} = 0$. When $\overline{K} = L(G_c)$, TC plays no effect and the second one is prohibitive.

For test controller implementation, we can list TC (s) for all $s\in L(TC/G_c)$ based on definition 3. Usually, this approach is impractical. So, we need a convenient implementation approach. Considering that we use an automaton to model the system under test, then using an automaton to describe the test controller TC may be a good manner. Specially, in this way the test controller will be finite and implementable. This automaton implementation of the test controller TC is called a realization of TC.

Definition 4: A realization for the test controller TC is a pair:

$$TC = (R, \phi) \tag{6}$$

where, $R = (X, \Omega, g, x_0, X_m)$ is a determinate finite state automaton and it can perform all behaviors defined by test cases; ϕ is a set of test control policies. Each element in ϕ is a test control policy that can perform certain test case. ϕ is defined on the state of R, ϕ: $X\rightarrow\Gamma$. Thus $\forall\Gamma\in\phi$ maps the test controller state $x\in X$ into $\{0, 1\}^{\Sigma_c}$.

Based on the definitions of Prefix-closure (Cassandras and Lafortune, 1999) and Controllability (Ramadge and Wonham, 1987), we present the existence condition for test controller:

Existence condition: Considering the system $G_c = (Q, q_0, Q_m, \Gamma\times\Sigma, \Lambda, \delta_c, \lambda_c)$, where $\Omega = \Sigma\cup\Lambda$ is the event set containing both input event and output event; $\Omega_{uc}\subseteq\Omega$ is the set of uncontrollable events and $\Lambda\subseteq\Omega_{uc}$. Let the test case set be a language $K\subseteq L(G_c)$, where $K\neq\varnothing$. Then there exists test controller TC such that $L(TC/G_c) = ???$ if and only if:

$$\begin{cases}(1)\ K\in L(M)\\ (2)\ \overline{K}\Omega_{uc}\cap L(M)\subseteq\overline{K}\end{cases}$$

The proof process of the existence condition is similar with proof in [15]. If the existence condition is satisfied, the test controller that can execute all test cases is:

$$TC(s) = \{\sigma\in\Sigma_c: (s\sigma\in\overline{K})\wedge(\sigma\subset H(q_0, s)\}$$

For constructing an automaton realization of test controller TC, we need to build an automaton "R" which marks the language $\overline{K}$. Its definition is:

$$R: = (X, \Omega, g, x_0, X_m) \tag{7}$$

Let we assume that R is trim and:

$$L_m(R) = L(R) = \overline{K} \tag{8}$$

The realization R can be seen as a device that can execute state transitions in response to input string $\omega\in\Omega^*$. G_c affect R in an event feedback manner by executing the state transitions of R according to the output of G_c. Then the following behavior of G_c is constrained by the control patterns defined on current state of R. The behavior of this closed-loop testing system is described by the product automaton of G_c and R, denoted by R/G_c,

$$R/G_c: = (X\times Q, \Omega, g\times\delta_c, (x_0, q_0), X_m\times Q_m) \tag{9}$$

where, H means cartesian product, (x_0, q_0) is the initial state of $R\times G_c$. The state transition function is formally defined as: $g\times\delta_c$: $\Omega\times X\times Q\rightarrow X\times Q$. Since R and G_c have the same event set Ω, then we get:

$$\begin{aligned}&L(R\times G_c) = L(R)\cap L(G_c) = \overline{K}\cap L(G_c) = \overline{K} = L(TC/G_c)\\ &L_m(R\times G_c) = L_m(R)\cap L_m(G_c) = \overline{K}\cap L_m(G_c)\\ &\qquad = L(TC/G_c)\cap L_m(G_c) = L_m(TC/G_c)\end{aligned} \tag{10}$$

$R\times G_c$ is a composition of two automata when performing reference to a test control policy. This test control paradigm can be interpreted as follow in the algebraic framework of parallel composition of automata. Let G_c and R be in state q and state x, respectively after the execution of string $s\in L(TC/G_c)$. σ is the enabled event generated by G_c currently. And it is also one of active events when R is in state x. R can be seen as a passive observer of G_c. Following the event σ, q' and x' are new states of two automata. Then the active event set of R at x' give enabled events set of G_c.

Test strategy: According to the above discussion, one test case can be formally defined by a string $tc \in L(G_c)$. For every test case, unique test control policy Γ_i is defined on the controlled input event of system. The execution of test case can be seen as controlling the system behavior under this control policy. When this policy is applied, testing is started. The whole testing process is implemented by switching the control policies through a sequence of elements $\Gamma_1, \Gamma_2, \Gamma_3, \ldots$ in ϕ.

In the process of system testing, test control policies are selected in accordance with a given testing strategy. A testing strategy determines what or which test case should be applied and when system testing should be stopped. The U-method is a fixed testing strategy: The test case is executed at a fix order one by one.

EXAMPLE

In this section, we will examine our automatic testing framework on test of Automatic Train Protection system (ATP), when U-method is used.

Mealy automaton model of ATP: ATP is a system that relays signal information, track speed information and other track information to trains and can automatically slow or stop trains if they exceed the track speeds or approach signals at STOP at too high a speed. (Railsafe, 2011) Here, we use the ATP in China Train Control System Level 3 (CTCS-3) for example. The ATP for CTCS-3 mainly consists of the 7 operation modes, which are full supervision mode (FS), shunting mode (SH), On-sight mode (OS), staff responsible mode (SR), isolation mode (IS), standby mode (SB) and sleeping mode (SL). We construct the automata model for ATP shown in Figure 2 and its states and events are described in Table 1-3.

Generated test cases: All UIO sequences for every state are shown in Table 4 according to the U-method.

Test controller for ATP based on U-method: ased on the test cases shown in Table 4, we construct one automatic test controller TC = (R, ϕ) for ATP. The required languages K based on U-method is:

$$K = \{I_1O_1I_2O_2I_{11}O_5I_{13}O_2; I_1O_1I_3O_4I_{12}O_5I_{13}O_2; I_1O_1I_4O_5I_{13}O_2I_{11}O_5;$$
$$I_1O_1I_5O_3I_6O_1I_2O_2; I_1O_1I_7O_6I_8O_1I_2O_2; I_1O_1I_2O_2I_9O_4I_{12}O_5;$$
$$I_1O_1I_3O_4I_{10}O_2I_{11}O_5; I_1O_1I_4O_5I_9O_4I_{12}O_5; I_1O_1I_3O_4I_7O_6I_8O_1;$$
$$I_1O_1I_2O_2I_7O_6I_8O_1; I_1O_1I_4O_5I_7O_6I_8O_1; I_1O_1I_5O_3I_7O_6I_8O_1;$$
$$I_1O_1I_4O_5I_5O_3I_6O_1; I_1O_1I_3O_4I_5O_3I_6O_1; I_1O_1I_2O_2I_5O_3I_6O_1\}$$

Table 1: States of ATP model

State	Description
s_0	Power off
s_1	Standby mode
s_2	Full supervision mode
s_3	Shunting mode
s_4	On-sight mode
s_5	Staff responsible mode
s_6	Isolation mode

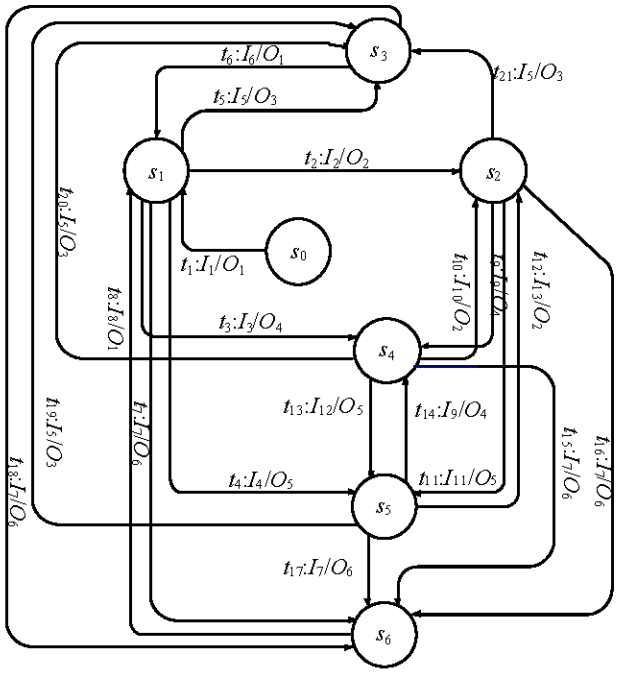

Fig. 2: Automata model of ATP

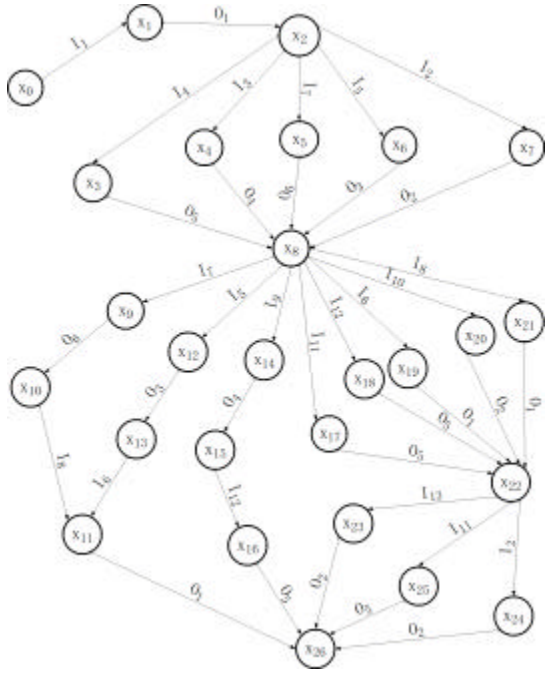

Fig. 3: Test controller for ATP

Table 2: Input events of ATP model

Input event	Number
I_0	Power off
I_1	Power on
I_2	Driver presses on button, ATP calls RBC successfully and position of train is valid
I_3	Driver presses on button, having on-sight route and the route is valid
I_4	Driver presses on button, no route and the driver presses on staff responsible button
I_5	Driver presses on shunting button
I_6	Driver presses on stopping button
I_7	Driver presses on isolation button
I_8	Driver turns isolation button to normal position
I_9	Having route and driver presses on on-sight button
I_{10}	Train receives a go-ahead signal
I_{11}	No route and driver selects overtaking
I_{12}	Driver selects overtaking
I_{13}	Train receives a go-ahead signal and position is not affirmed

Table 3: Output events of ATP model

Input event	Number
O_0	DMI shows power off
O_1	DMI shows standby mode
O_2	DMI shows full supervision mode
O_3	DMI shows shunting mode
O_4	DMI shows on-sight mode
O_5	DMI shows staff responsible mode
O_6	DMI shows isolation mode

Table 4: UIO sequences for each state

State	UIO sequences
s_0	$[t_1: I_1/O_1]$
s_1	$[t_2: I_2/O_2]$; $[t_3: I_3/O_3]$; $[t_4: I_4/O_4]$
s_2	$[t_{11}: I_{11}/O_5]$
s_3	$[t_6: I_6/O_1]$
s_4	$[t_{10}: I_{10}/O_2]$; $[t_{13}: I_{12}/O_5]$
s_5	$[t_{12}: I_{13}/O_2]$
s_6	$[t_8: I_8/O_1]$

By the checking the state transition structure of ATP model, K fulfill the test controller existence condition $\overline{K}\Omega_{uc}\cap L(M)\subseteq\overline{K}$ and $K\in L(M)$. One realization R of test controller is shown in Fig. 3. It has the same event set with the automaton model of ATP. Because of the length of study, we will not show the test control policy set here. For example, the test control policy for test case t_1, t_2, t_{11}, t_{12} is:

$$\{\Gamma_1(x_0)(I_1)=1, \Gamma_2(x_2)(I_2)=1, \Gamma_1(x_8)(I_{11})=1,$$
$$\Gamma_1(x_{22})(I1_{3})=1\}$$

CONCLUSION

In this study, because of lack of automatic testing system model, we proposed a novel automatic testing system model based on controlled automata. Specially, we set up the theory framework of automatic testing based on event feedback. The test controller, system under test and test observer compose a close-loop testing system based on event feedback. The system under test modeled by controlled automata serves as the controlled object. Meanwhile we gave the existence condition and

construction method of the test controller, which is implemented as a pair: TC = (R, ϕ).

In a word, we set up a novel automatic testing framework for man-made system. The example of ATP testing shows the feasibility of our method.

ACKNOWLEDGMENT

This study was supported by the NSFC under Grants 61104019 and 61004070, the National Key Technology R and D Program under Grant 2009 BAG12A08 and Tsinghua University Initiative Scientific Research Program.

REFERENCES

Chen, T.Y. and Y.T. Yu, 1996. On the expected number of failures detected by subdomain testing and random testing. IEEE Trans. Software Eng., 22: 109-119.

Chen, T.Y. and Y.T. Yu, 2002. A decision-theoretic approach to the test allocation problem in partition testing. IEEE Trans. Syst. Man Cybemetics-Part A: Syst. Humans, 32: 733-745.

Cai, K.Y., 2002. Optimal software testing and adaptive software testing in the context of software cybernetics. Inform. Software Technol., 44: 841-855.

Cassandras, C.G. and S. Lafortune, 1999. Introduction to Discrete Event Systems. Kluwer Academic Publishers, Dordrecht, Netherlands.

Gao, Z.H., X.H. Chen, D.L. Peng, X.K. Liu, X.Q. Wang and F.Y. Zheng, 2010. Dynamic automatic testing and calibration system of time grating sensor. Instrument Tech. Sensor, 2: 95-97.

Gutjahr, W.J., 1999. Partition testing vs. random testing: The influence of uncertainty. IEEE Trans. Software Eng., 25: 661-674.

Hopscroft, J.E., R. Motwani and J.D. Ullman, 2000. Introduction to Automata Theory, Languages and Computation. 2nd Edn., Addison Wesley Co., USA.

Lee, D. and M. Yannakakis, 1996. Principles and methods of testing finite state machines-a survey. Proc. IEEE, 84: 1090-1123.

Liu, S.Y. and S. Nakajima, 2011. A framework for automatic functional testing based on formal specifications. Proceedings of the 6th International Workshop on Automation of Software Test, May 21-28, 2011, Honolulu, HI, USA., pp: 107-108.

Meyer, B., I. Ciupa, A. Leitner and L.L. Liu, 2007. Automatic testing of object-oriented software. Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science, January 20-26, 2007, Harrachov, Czech Republic, pp: 114-129.

Poore, J.H., 2000. Introduction to the special issue on: Model-based statistical testing of software intensive systems. Inform. Software Technol., 42: 797-799.

Ramadge, P.J. and W.M. Wonham, 1987. Supervisory control of a class of discrete event processes. SIAM J. Control Optimizat., 25: 206-230.

Railsafe, 2011. Automatic train protection. https://railsafe.org.au/automatic-train-protection

Sahinoglu, M., 2003. An empirical Bayesian stopping rule in testing and verification of behavioral models. IEEE Trans. Instrumentation Measurement, 52: 1428-1443.

Sabnani, K. and A. Dahbura, 1988. A protocol test generation procedure. Comput. Networks ISDN Syst., 15: 285-297.

Tal, O., C. McCollin and A. Bendell, 2002. An optimal statistical testing policy for software reliability demonstration of safety-critical systems. Eur. J. Operat. Res., 137: 544-557.

Whittaker, J.A., 2000. What is software testing? And why is it so hard? IEEE Software, 17: 70-79.