Incremental Model-based Test Suite Reduction with Formal Concept Analysis

Pin Ng*, Richard Y. K. Fung** and Ray W. M. Kong***

Abstract—Test scenarios can be derived based on some system models for requirements validation purposes. Model-based test suite reduction aims to provide a smaller set of test scenarios which can preserve the original test coverage with respect to some testing criteria. We are proposing to apply Formal Concept Analysis (FCA) in analyzing the association between a set of test scenarios and a set of transitions specified in a state machine model. By utilizing the properties of concept lattice, we are able to determine incrementally a minimal set of test scenarios with adequate test coverage.

Keywords—Test Suite Reduction, Model-based Testing, State Machine Model, Formal Concept Analysis

1. INTRODUCTION

Test scenarios can be derived based on some system models for requirements validation purposes. Model-based testing [1, 16] refers to deriving a suite of test scenarios from a model that represents the behavior of a software system. In particular, state machine model has been widely used for this purpose in testing event-driven, reactive systems, and embedded software systems [1, 2]. State machine model can be used to specify the dynamic perspective of a system and its interactions with the users through sequences of transitions. The sequences of transitions can form a set of test scenarios for validation of functional requirements by test engineers and end users. However, since cycles in the state machine model may lead to an infinite number of test scenarios, exhaustive testing is usually not possible. Moreover, many test scenarios are part of some other test scenarios and thus lead to redundancy in the test suite. Model-based test suite reduction can be applied in this situation and derive a smaller set of test scenarios which still preserves the original test coverage with respect to some testing criteria. A default criterion of adequate testing with a state machine model is all-transition coverage criterion [1, 11, 16], which means each transition specified in the state machine model should be triggered at least once by executing the test scenarios.

In this paper, we shall describe an incremental approach for reducing model-based test suite using Formal Concept Analysis (FCA) [6]. FCA is a mathematical technique for formulating concepts in terms of a set of formal objects and their associated formal attributes, and providing

Manuscript received March 15, 2010; accepted April 20, 2010.
Corresponding Author: Pin Ng
* Hong Kong Community College, Hong Kong Polytechnic University, Hong Kong (ccpng@hkcc-polyu.edu.hk)
** Dept. of Manufacturing Engineering and Engineering Management, City University of Hong Kong, Hong Kong. (richard.fung@cityu.edu.hk)
*** Automatic Manufacturing Limited, Hong Kong (raykong@automatic.com.hk)

a systematic way of combining and organizing individual concepts of a given context into hierarchically ordered conceptual structure, known as a concept lattice. In the context of transition coverage, FCA can be applied to associate a set of test scenarios (as formal objects) with a set of transitions (as formal attributes) specified in a state machine model, and to organize them to form a concept lattice. By utilizing the properties of concept lattice, we are able to incrementally determine a minimal set of test scenarios with adequate test coverage.

This paper is organized as follows. Section 2 discusses some related work. Section 3 presents the application of FCA in test suite reduction. The proposed incremental approach for model-based test suite reduction is explained in Section 4. Finally, Section 5 concludes our work.

2. RELATED WORK

Test suite reduction, in general, can be considered as a minimum set-covering problem [4]. A classical approach for solving minimum set-covering problem is based on greedy heuristic [5]. When applying greedy heuristic for test scenario selection, first, the test scenario that covers the most elements will be selected. Then, the test scenario that covers the most remaining elements will be selected. The process will be repeated until all the elements have been covered. In case there are multiple test scenarios covering the most and same amount of elements, one of the test scenarios will arbitrarily selected. However, greedy heuristic may not always provide the optimal test suite [12, 14].

For example, Table 1 shows the coverage of six transitions {t1, t2, t3, t4, t5, t6} by four test scenarios {s1, s2, s3, s4}. Suppose that we would like to determine a reduced set of test scenarios that can sufficiently cover all the transitions. When applying greedy heuristic, s1 will be selected first for having the greatest coverage cardinality. Then s2 and s3 will also be selected for covering the remaining transitions. Therefore, the greedy heuristic will derive a reduced test suite of {s1, s2, s3}. However, the minimal test suite for this simple case, in fact, is {s2, s3}. This example reveals a limitation of applying greedy heuristic in test suite reduction. With greedy heuristic, we could have selected some test scenarios which may turn out to be redundant when some other test scenarios are included in the test suite. Our proposed incremental approach, which is based on FCA, can help to identify those redundant test scenarios so as to keep the test suite minimal.

FCA has been applied to several software engineering problems [15], such as restructuring program codes into more cohesive components, identifying class candidates in object oriented design, and re-engineering class hierarchies. Most of such work applies FCA to model the generalization-specialization relationship, in which, a subclass inherits some features from its superclasses within a class hierarchy; and to model the variables dependency relationship for de-

Table 1. A simple case

R	t1	t2	t3	t4	t5	t6
s1	x	x	x	x		
s2	x	x			x	
s3			x	x		x
s4	x	x				

sign recovery. Our approach makes use of the concept analysis mechanism to support incremental reduction of model-based test suite with reference to state machine model, which is widely used to model the behavioral perspective of software systems.

Having been inspired by FCA, Tallam and Gupta [14] presented a Delayed-Greedy heuristic for selecting the minimum number of test cases which can exercise the given set of testing requirements. Our mechanism differs from Tallam and Gupta's approach in which, our testing requirements are based on the test scenarios derived from state machine model; and our approach does not need to go through attribute reduction procedure as described in their Delayed-Greedy algorithm. Because of the involvement of attribute reduction procedure, Tallam and Gupta's approach cannot support incremental update of the test suite in the situations that when some new test cases have been incurred.

Sampath et al. [13] have applied FCA for test suite reduction in the domain of web applications testing, in which, each of the URLs used in a web session is considered as a formal attribute; whilst each web session is considered as a formal object which constitutes to be a test case. The reduced test suite is obtained by selecting those test cases associated with the strongest concepts (i.e. the concept nodes that are just above the bottom-most concept node in the concept lattice). Although the method is able to support incremental selection of test cases, redundancy may still exist among the strongest concepts and thus the reduced test suite may not be minimal. By utilizing the incremental mechanism for updating the concept lattice structure, our approach can iteratively locate for any test scenarios which turn out to be non-significant or redundant when new test scenarios are added. These non-significant or redundant test scenarios will be removed in order to keep the test suite minimal.

3. APPLYING FCA IN TEST SUITE REDUCTION

Formal Concept Analysis (FCA) provides a mathematical foundation for systematically combining and organizing individual concepts of a given context into a hierarchically ordered conceptual structure [6]. Given a binary relation R between a set of *formal objects* O and a set of *formal attributes* A (that is, $R \subseteq O \times A$), the tuple (O, A, R) forms a *formal context*. For a set of objects, $O_i \subseteq O$, the set of *common attributes*, σ, is defined as:

$$\sigma(O_i) = \{\ a \in A \mid \forall(o \in O_i)\ (o, a) \in R\ \} \tag{1}$$

Analogously, the set of *common objects*, τ, for a set of attributes, $A_i \subseteq A$, is defined as:

$$\tau(A_i) = \{\ o \in O \mid \forall(\ a \in A_i)\ (o, a) \in R\ \} \tag{2}$$

With reference to equations (1) and (2), a *concept* c can be defined as an ordered pair (O_i, A_i) such that $A_i = \sigma(O_i)$ and $O_i = \tau(A_i)$. That means, all and only objects in O_i share all and only attributes in A_i. For a concept $c = (O_i, A_i)$, O_i is called the extent of c, denoted by $Extent(c)$, and A_i is called the intent of c, denoted by $Intent(c)$. The set of all concepts of a given formal context forms a partial order by:

$$c_1 \le c_2 \Leftrightarrow Extent(c_1) \subseteq Extent(c_2);\ \text{or equivalently,}\ \ c_1 \le c_2 \Leftrightarrow Intent(c_1) \supseteq Intent(c_2). \tag{3}$$

The partial order relation in equation (3) can be used to specify the meanings of subconcepts and superconcepts. Given two concepts c_1 and c_2, if $c_1 \leq c_2$ holds, c_1 is called *subconcept* of c_2; or equivalently, c_2 is called *superconcept* of c_1.

The set of all concepts of a formal context and the partial ordering can be represented graphically using a *concept lattice*. A concept lattice consists of nodes that represent the concepts and edges connecting these nodes. The nodes for concepts c_1 and c_2 are connected if and only if $c_1 \leq c_2$ and there is no other concept c_3 such that $c_1 \leq c_3 \leq c_2$.

When applying FCA in model-based test suite reduction, the formal context for transition coverage [9, 10] can be defined as a tuple (*S*, *T*, *R*), where:

- *S* is a set of *test scenarios* (considered as formal objects);
- *T* is a set of *transitions* (considered as formal attributes) that appear in the given state machine model;
- a pair (scenario *s*, transition *t*) is in relation *R* if transition *t* is triggered when scenario *s* is executed.

As an example, with reference to the transition coverage of the test scenarios given in Table 1, a set of seven concepts that can be derived is shown in Table 2.

Fig. 1 depicts the concept lattice for the concepts listed in Table 2. Each concept node is labeled with the associated extent and intent elements. The Top concept, c1, of the concept lattice is the most generalized concept – the superconcept to all other concepts; whereas the Bottom concept, c7, is the most specialized concept – the subconcept to all other concepts.

The labeling of the lattice can be simplified for clarity by applying equations (4) and (5) so that only the extent and intent elements which are most specific to a given concept are displayed.

$$AttributeLabels(c) = Intent(c) - \bigcup_{\forall c_j \geq c} Intent(c_j) \tag{4}$$

$$ObjectLabels(c) = Extent(c) - \bigcup_{\forall c_i \leq c} Extent(c_i) \tag{5}$$

The attribute labels are displayed slightly above the concept node whereas the object labels are marked slightly below the node. Fig. 2 shows the concept lattice with a compact form of labeling. The concept lattice structure can be used for checking the adequacy of test coverage and determining a minimal set of test scenarios.

Table 2. List of concepts based on the simple case

Concept	*Extent ( )*	*Intent ( )*
c1	{s1, s2, s3, s4}	{ }
c2	{s1, s2, s4}	{t1, t2}
c3	{s1, s3}	{t3, t4}
c4	{s1}	{t1, t2, t3, t4}
c5	{s2}	{t1, t2, t5}
c6	{s3}	{t3, t4, t6}
c7	{ }	{t1, t2, t3, t4, t5, t6}

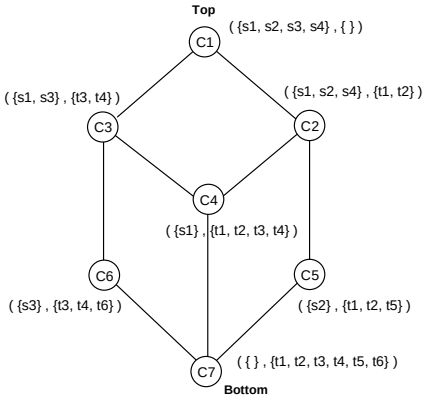

Fig. 1. Concept lattice with full labeling

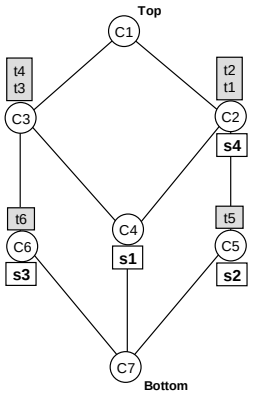

Fig. 2. Concept lattice in compact form

3.1 Adequacy of Test Coverage

In the context of transition coverage, the set of test scenarios is considered to be providing adequate test coverage if when executing these test scenarios, each transition is triggered at least once. With reference to the concept lattice, the adequacy of test coverage is indicated by:

$$\textit{AttributeLabels}(\text{Bottom}) = \varnothing \wedge \textit{ObjectLabels}(\text{Bottom}) = \varnothing \tag{6}$$

The condition in equation (6) implies that every transition is covered by some test scenarios. Therefore, the concept lattice shown in Fig.2 reveals that the test scenarios {s1, s2, s3, s4} are sufficient enough to cover all transitions.

3.2 Minimal Set of Test Scenarios

A set of test scenarios is considered to be minimal if any one of the test scenarios is removed, some of the transitions will not be covered by the remaining test scenarios. The concept lattice structure can help in determining which test scenarios can be excluded from the test suite without affecting the test coverage.

Definition 1: Non-significant test scenario

With reference to a concept lattice, a test scenario *s* is *non-significant* if:

(i) $s \in \textit{ObjectLabels}\ (c)$; and

(ii) there exists some concept c' such that

$c \geq c' \geq \text{Bottom}$

A test scenario *s* is *non-significant* implies that its coverage of transitions is a subset to that of at least one of the other test scenarios.

Definition 2: Redundant test scenario

With reference to a concept lattice, a test scenario *s* is *redundant* if:

(i) $s \in \textit{ObjectLabels}\ (c)$; and

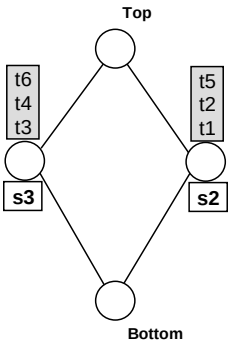

Fig. 3. Updated concept lattice for the minimized test suite {s2, s3}

(ii) there is no other concept c' such that
$c \geq c' \geq$ Bottom; and
(iii) $AttributeLabels(c) = \varnothing$

A test scenario s is *redundant* implies that it does not solely cover any transitions by itself. With reference to the example shown in Fig. 2, by Definition 1, test scenario s4 is considered to be non-significant; whereas, by Definition 2, s1 is considered to be redundant. The test suite can be reduced whilst maintaining the adequacy of test coverage by removing those non-significant test scenarios and redundant test scenarios. For example, Fig. 3 shows the revised concept lattice after removing the test scenarios s4 and s1. The resultant test suite {s2, s3} is considered to be minimal.

4. INCREMENTAL MODEL-BASED TEST SUITE REDUCTION

The software system may evolve as requirements change and thus lead to the need for additional test scenarios to be considered. In this section, we shall describe the incremental mechanism that can support the incremental updates of the test suite.

The existence of incremental algorithms for updating concept lattices in the literature [3, 7] make it possible to save the effort of reconstructing the whole lattice from scratch. Fig. 4 shows the algorithm which built upon the incremental lattice update mechanism, including adding new objects and removing existing objects [3, 7]. The algorithm starts with a set of test scenarios S with an initial concept lattice L. Each test scenario is added to the concept lattice one by one. If the test scenario turns out to be non-significant, it will be removed. In case some redundant test scenarios exist in the updated concept lattice, they will also be removed from the lattice structure in order to keep the test suite minimal. The process will be repeated until all test scenarios in S have been considered. The output of the algorithm will be a set S' which contains the minimal set of test scenarios and the corresponding updated concept lattice L'.

As a working example, we demonstrate the incremental model-based test suite reduction process with a case of Automated Teller Machine (ATM). ATM is a commonly used example in the literature for explaining the modeling with state machine model [1, 8, 16]. Fig. 5 illustrates a state machine model of a simplified ATM system. It models the interactions between a user and the ATM. First, the ATM will perform user authentication by checking the validity of the ATM

```
Algorithm: incremental selection of test scenarios

Input:   S = {s1, s2, … sn}, a set of test scenarios
         L, an initial concept lattice

Output:  S′, a minimized subset of S,
             with the same test coverage of S
         L′, an updated concept lattice containing
             the elements of S′

procedure selectTestScenarios(S, L)
begin
   S′ := ∅
   L′ := L
   for each si ∈S do
      /* add new test scenario to the concept lattice */
      addObject(L′, si )
      S′ := S′ ∪ {si}

      /* check for non-significant test scenario */
      for each sj ∈ S′ do
         if sj is non-significant     /* see Definition 1 */
            /* remove non-significant test scenario */
            removeObject(L′, sj )
            S′ := S′ \ {sj}
         endif
      endfor
      /* check for redundant test scenario */
      for each sj ∈ S′ do
         if sj is redundant           /* see Definition 2 */
            /* remove redundant test scenario */
            removeObject(L′, sj )
            S′ := S′ \ {sj}
         endif
      endfor
   endfor
   return S′, L′
end
```

Fig. 4. Incremental selection of test scenarios

card and password. Then, the user is allowed to choose the services for balance checking, money withdrawal, or fund transfer, given that the user has sufficient balance in the bank account. In validating the ATM system, a series of test scenarios will be applied in order to check whether the ATM can perform according to the requirements specified in the state machine model. Each test scenario will trigger a sequence of transitions which will cause changes in the states of the system. For instance, the scenario of a valid money withdrawal will trigger the following sequence of transitions: t01→t02→ t04→ t08→ t012→ t15→ t16.

By traversing the state machine model, we can trace for a collection of feasible sequences of transitions which forms a set of test scenarios as listed in Fig. 6. We can then apply the incremental algorithm to consider each test scenario one by one so as to determine a minimal set of

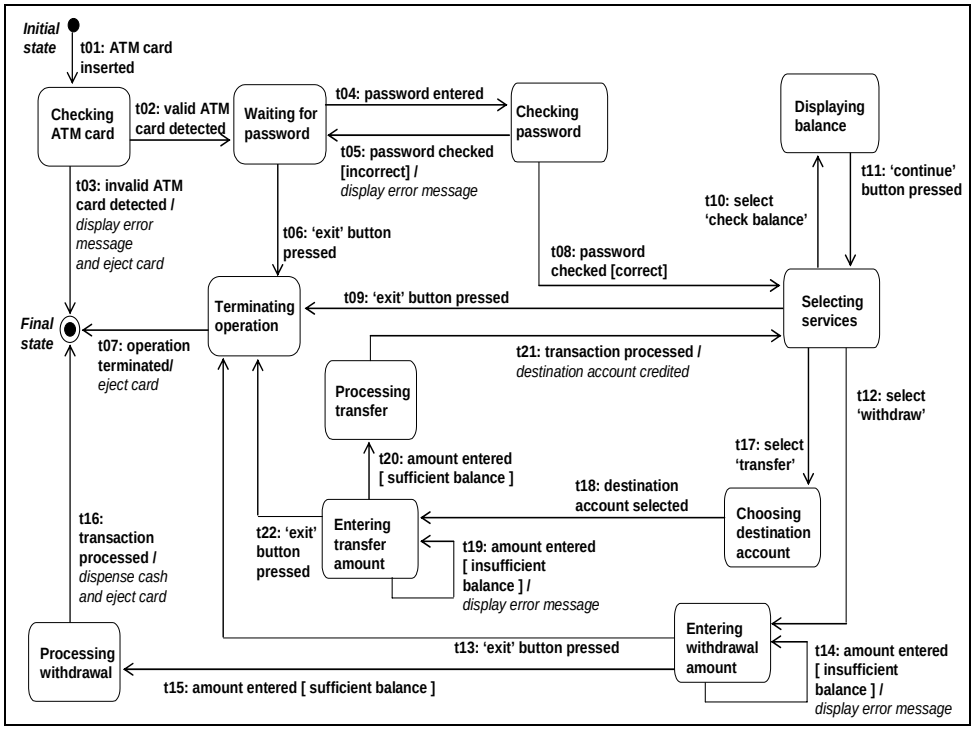

Fig. 5. State machine model of an ATM system

Scenario s01: t01→ t03
Scenario s02: t01→ t02→ t06→ t07
Scenario s03: t01→ t02→ t04→ t05→ t06→ t07
Scenario s04: t01→ t02→ t04→ t08→ t09→ t07
Scenario s05: t01→ t02→ t04→ t08→ t10→ t11→ t09→ t07
Scenario s06: t01→ t02→ t04→ t08→ t10→ t11→ t12→ t13→ t07
Scenario s07: t01→ t02→ t04→ t08→ t12→ t13→ t07
Scenario s08: t01→ t02→ t04→ t08→ t12→ t14→ t13→ t07
Scenario s09: t01→ t02→ t04→ t08→ t12→ t15→ t16
Scenario s10: t01→ t02→ t04→ t08→ t10→ t11→ t17→ t18→ t22→ t07
Scenario s11: t01→ t02→ t04→ t08→ t17→ t18→ t22→ t07
Scenario s12: t01→ t02→ t04→ t08→ t17→ t18→ t19→ t22→ t07
Scenario s13: t01→ t02→ t04→ t08→ t17→ t18→ t20→ t21→ t09→ t07

Fig. 6. List of test scenarios

test scenarios.

With the inclusion of a new test scenario in each iteration, the concept lattice can help to check whether such incremental update would lead to any existing test scenarios becoming non-significant or redundant. If so, those non-significant or redundant test scenarios should be excluded from the set *S′*. Table 3 shows the inclusion and exclusion of test scenarios in each iteration for the working example. The resultant concept lattice is shown in Fig. 7, which indicates

Table 3. Inclusion and exclusion of test scenarios in *S′*

Iteration	Add scenario	Remove scenario	Reason for removal	The resultant *S′*
1	s01	-		{s01}
2	s02	-		{s01, s02}
3	s03	s02	s02 is non-significant	{s01, s03}
4	s04	-		{s01, s03, s04}
5	s05	s04	s04 is non-significant	{s01, s03, s05}
6	s06	-		{s01, s03, s05, s06}
7	s07	s07	s07 is non-significant	{s01, s03, s05, s06}
8	s08	s06	s06 is redundant	{s01, s03, s05, s08}
9	s09	-		{s01, s03, s05, s08, s09}
10	s10	-		{s01, s03, s05, s08, s09, s10}
11	s11	s11	s11 is non-significant	{s01, s03, s05, s08, s09, s10}
12	s12	s10	s10 is redundant	{s01, s03, s05, s08, s09, s12}
13	s13	-		{s01, s03, s05, s08, s09, s12, s13}

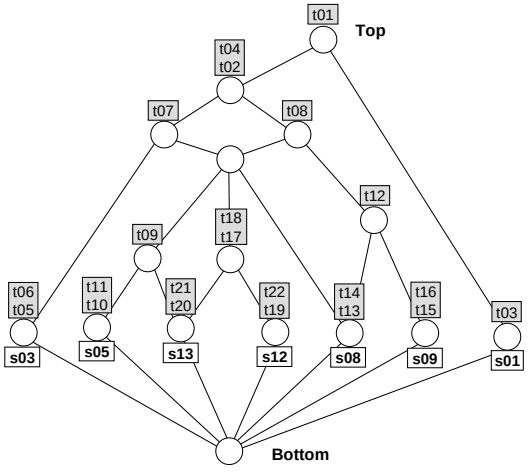

Fig. 7. List of test scenarios

that, among the given test scenarios, seven of them are selected to form the minimal set of test suite {s01, s03, s05, s08, s09, s12, s13}. By executing this set of test suite, all the transitions specified in the state machine model will be triggered at least once. Some narrative texts can be added as shown in Table 4 for completing the specification of the selected test scenarios. Later

Table 4. Specification of the selected test scenarios

Selected Test Scenarios	Sequences of Transitions	Expect Results
Scenario s01: "Invalid ATM card"	t01: ATM card inserted→ t03: invalid ATM card detected	Warning message displayed and ATM card ejected without any transaction.
Scenario s03: "Incorrect password"	t01: ATM card inserted → t02: valid ATM card detected → t04: password entered → t05: password checked [incorrect] → t06: 'exit' button pressed → t07: operation terminated	ATM card ejected without any transaction.
Scenario s05: "checking balance"	t01: ATM card inserted → t02: valid ATM card detected → t04: password entered → t08: password checked [correct] → t10: select 'check balance' → t11: 'continue' button pressed → t09: 'exit' button pressed → t07: operation terminated	Correct balance amount displayed and ATM card ejected.
Scenario s08: "withdrawal with insufficient balance"	t01: ATM card inserted → t02: valid ATM card detected → t04: password entered → t08: password checked [correct] → t12: select 'withdraw' → t14: amount entered [insufficient balance] → t13: 'exit' button pressed → t07: operation terminated	ATM card ejected without any transaction.
Scenario s09: "withdrawal with sufficient balance"	t01: ATM card inserted → t02: valid ATM card detected → t04: password entered → t08: password checked [correct] → t12: select 'withdraw' → t15: amount entered [sufficient balance] → t16: transaction processed	Correct amount of cash dispensed and ATM card ejected.
Scenario s12: "transfer with insufficient balance"	t01: ATM card inserted → t02: valid ATM card detected → t04: password entered → t08: password checked [correct] → t17: select 'transfer' → t18: destination account selected → t19: amount entered [insufficient balance] → t22: 'exit' button pressed → t07: operation terminated	ATM card ejected without any transaction.
Scenario s13: "transfer with sufficient balance"	t01: ATM card inserted → t02: valid ATM card detected → t04: password entered → t08: password checked [correct] → t17: select 'transfer' → t18: destination account selected → t20: amount entered [sufficient balance] → t21: transaction processed → t09: 'exit' button pressed → t07: operation terminated	Correct amount of money transferred to the destination account and ATM card ejected.

on, if the software requirements of the ATM system evolve, extra test scenarios can be derived from the revised state machine model. The set of new test scenarios will be used as the input to the incremental algorithm for updating the concept lattice and determining the revised minimal test suite.

5. CONCLUSION

FCA provides a mathematical foundation for combining and organizing individual concepts of a given context to form a concept lattice. In this paper, we have applied FCA in incremental model-based test suite reduction, with reference to the behavioral perspective of a system – state machine model, through analyzing the transition coverage relationship of " test scenario *s* covers transition *t*" .

Software system may evolve as requirements change and thus lead to the need for additional test scenarios to be considered. With the notion of concept lattice, the primary contributions of this work are: (1) utilizing the properties of concept lattice in selecting a minimal set of test scenarios while maintaining adequate test coverage; and (2) supporting incremental update of the minimal test suite to cater for the evolving software requirements.

REFERENCES

[1] R. V. Binder, Testing Object-Oriented Systems-Models, Patterns, and Tools, Object Technology. Addison-Wesley, 2000.

[2] B. Broekman and E. Notenboom, Testing Embedded Software, Addison-Wesley, 2003.
[3] C. Carpineto and G. Romano, Concept Data Analysis: Theory and Applications, Wiley, 2004.
[4] T.Y. Chen and M.F. Lau, "A New Heuristic for Test Suite Reduction," Information and Software Technology, 40, 1998, pp.347-354.
[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, MIT Press, Second Edition, 2001.
[6] B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, 1999.
[7] R. Godin, R. Missaoui, and H. Alaoui, "Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices," Computational Intelligence, 11(2), 1995, pp.246-267.
[8] P.V.R. Murthy, P.C. Anitha, M. Mahesh, and R. Subramanyan, " Test ready UML statechart models," Proceedings of the 2006 international workshop on scenarios and state machines: models, algorithms, and tools, SCESM '06, May, 2006, pp.75-81.
[9] P. Ng and R.Y.K. Fung, "Applying Formal Concept Analysis in Requirements Validation with UML State Machine Model," International Journal of Computer & Information Science, Vol. 8, No. 4, December 2007, pp.550-559.
[10] P. Ng and R.Y.K. Fung, "Model-Based Test Suite Reduction with Concept Lattice," Proceedings of Advanced Software Engineering and Its Applications, ASEA 2008, Dec., 2008, pp.3-8.
[11] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, "Generating Test Data from State-based Specifications," Software Testing, Verification and Reliability, Vol.13, Iss. 1, 2003, pp.25-53.
[12] G. Rothermel, R. Untch, C. Chu, and M.J. Harrold, " Prioritizing Test Cases for Regression Testing," IEEE Transactions of Software Engineering, Vol. 27, No. 10, Oct., 2001, pp.929-948.
[13] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock, "A Scalable Approach to User-Session based Testing of Web Applications through Concept Analysis," Proceedings of 19th International Conference on Automated Software Engineering, ASE '04, Linz, Austria, 2004, pp.132-141.
[14] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," The 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, PASTE '05, 2005, pp.35-42.
[15] T. Tilley, R. Cole, P. Becker, and P. Eklund, "A Survey of Formal Concept Analysis Support for Software Engineering Activities," Formal Concept Analysis, LNAI 3626, B. Ganter et al. (Eds.), Springer-Verlag Berlin Heidelberg, 2005, pp.250-271.
[16] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.

Pin Ng

Pin Ng is a qualified CISA (Certified Information Systems Auditor) and currently a lecturer at the Hong Kong Community College - Hong Kong Polytechnic University. He received an MSc degree in Operational Research from University of Warwick, UK. and M. Phil. in Software Engineering from University of Hong Kong. He is currently pursuing an engineering doctorate degree, under the supervision of Dr. Richard Y. K. Fung, at City University of Hong Kong. His research focuses on the application of formal mechanisms in software requirements engineering.

Richard Y. K. Fung

Richard Y. K. Fung obtained a B.Sc.(Hons) degree in Production Engineering and a Master of Philosophy (M.Phil.) degree in Manufacturing Resource Planning, both from the Aston University in Birmingham, UK. Subsequently, he was awarded a Ph.D. degree in Customer Requirements Management by Loughborough University, UK. He worked in the industry for over twelve years, having been involved in different manufacturing areas including product development, production planning and control, design and implementation of management information systems, and management consultancy in the UK. Dr. Fung joined the City University of Hong Kong in 1989, and he is now an Associate Professor and the Director of the Laboratory of Enterprise Knowledge Integration and Transfer in the Department of Manufacturing Engineering and Engineering Management. His current scope of teaching and research covers Knowledge Management, Quality Management, Customer Requirements Analysis, Quality Function Deployment, Supply Chain and Logistics Management, Product Lifecycle Management, and the applications of Artificial Intelligence Techniques in the industry.

Ray W. M. Kong

Ray W. M. Kong is a qualified member of IEEE (Institute of Electrical and Electronics Engineering) for over eleven years. He obtained an MSc degree in Automation System and Management in the Department of Manufacturing Engineering and Engineering Management from City University of Hong Kong in 1998. Further, he was awarded an Engineering Doctorate degree in the City University of Hong Kong in 2008. He has worked at management level in the manufacturing and operation sectors in the electronics, toys, apparel and information technology industry for over twelve years. He is now an assistant general manager in Automatic Manufacturing Limited in Hong Kong.