Experience of Using Formal Concept Analysis in Supporting Software Testing Activities

Pin Ng

Hong Kong Community College, Hong Kong Polytechnic University, Hong Kong

Abstract
Formal concept analysis (FCA) provides a theoretical foundation for systematically arranging individual concepts of a given context into hierarchically ordered conceptual structure. The technique has been applied to solve several software engineering problems, such as restructuring program codes, identifying class candidates in object oriented design, and re-engineering class hierarchies. In relation to software testing, FCA can be used for determining a minimum number of test cases which can exercise the given set of test requirements. The FCA mechanism is particularly useful in supporting model-based software testing.
Keywords: *Formal Concept Analysis, Software Testing.*

1. Introduction

Formal Concept Analysis (FCA) is a mathematical technique for clustering objects that have common discrete attributes [4]. The technique formulates concepts in terms of objects and their associated attributes, and provides a systematic way of combining and organizing individual concepts of a given context into a concept lattice. FCA has been applied to several software engineering problems [10], such as restructuring the code into more cohesive components, identifying class candidates, locating features in the code by means of dynamic analysis, and reengineering class hierarchies

Software testing is an essential part of software development for the purposes of quality assurance, reliability estimation, and verification and validation. However, software testing is an extremely costly and time consuming process [5]. In the context of software testing, FCA can be applied to associate a set of test scenarios (as formal objects) with a set of test requirements (as formal attributes) and organize them to form a concept lattice. By analyzing the concept lattice structure, we can determine a minimal set of test scenarios with adequate test coverage. This could help to save the cost in test cases execution, and thus, reduce the cost of software development.

2. An Overview of FCA

FCA provide a systematic way for formulating concepts in terms of formal objects and their associated formal attributes [4]. With FCA, the individual concepts are organized and depicted in form of a hierarchically ordered conceptual structure, known as concept lattice. As a simple illustration, by considering a set of integers {1, 2, 3, 4} and a set of properties {odd, even, prime}, Table 1 shows a simple context table that defines the relationship between the set of integers and their associated properties.

Table 1: A simple context table

number	odd	even	prime
1	x		
2		x	x
3	x		x
4		x	

With the notion of FCA, a concept is an ordered pair formed by clustering a subset of formal objects (integers in the example) with a subset of formal attributes (properties in the example) that are commonly shared by the objects. For example, the ordered pair ({2, 3}, {prime}) forms a concept because the integers '2' and '3' commonly share the property 'prime'; or equivalently, the property 'prime' is valid to the integers '2' and '3' only. Based on the subset relation among the elements of the concepts, a concept lattice can be derived. For example, with reference to the context table in Table 1, the corresponding concept lattice is depicted in Figure 1.

The concept lattice can serve as a natural hierarchical ordering of concepts, in which the concepts at a higher level are considered as superconcept to those subconcepts at the lower part of the hierarchy. The "superconcept-subconcept" relation is useful in analyzing software

artifacts. For instance, in object-oriented software design, the "superconcept-subconcept" relation implies the inheritance relationship among the superclasses and sublcasses.

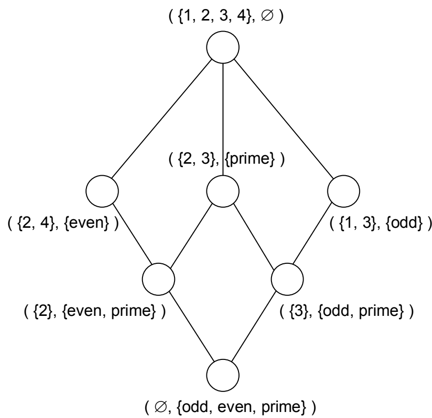

Fig.1 A concept lattice

The concept lattice structure is a useful tool for data analysis, knowledge discovery, and information retrieval. In the domain of software engineering, FCA has typically been applied to reengineering, refactoring and design recovery. Tilley et al. [10] did a survey and classified a broad collection of research work regarding the application of FCA in various activities of software engineering:

- *Requirement analysis*: Use cases are commonly used in requirements elicitation and analysis. By considering the use cases as formal objects and the nouns identified within the requirement text as formal attributes, the corresponding concept lattice forms the basis of a class hierarchy.

- *Component retrieval for software reuse*: FCA has been used as a formal mechanism in supporting the retrieval of software components from a software library. The software components are indexed by keywords based on FCA.

- *Formal specification*: With reference to the static structure of a formal specification, by considering each schema as a formal object and the individual mark-up elements as formal attributes, the formal specification can be navigated and explored visually with FCA.

- *Dynamic analysis*: By analyzing the dynamic aspects of software systems with FCA, specific parts of the software architecture related to use cases can be recovered.

- *Analyzing legacy system*: The general approach is to consider program functions and data structures as formal objects and formal attributes, respectively, for examining the configuration structure of legacy systems with FCA and then deriving object-oriented models from the legacy systems.

- *Reengineering class hierarchies*: FCA has been applied in reorganizing class hierarchies and recovering design patterns by considering a formal context where the formal objects are methods and the formal attributes are classes.

3. Software Testing with FCA

Software testing is an important activity in software development for facilitating quality assurance, reliability estimation, and verification and validation. However, software testing usually incurs high cost and time consumptions [5]. As a result, model-based testing was advocated for [1, 11] advocated for improving the efficiency and effectiveness of test cases generation. Model-based testing is a system testing technique that derives a suite of test cases from a system model representing the behavior of a software system. By executing the set of model-based test cases, the conformance of the target system to its specification can be validated.

One commonly used system model for model-based testing is state machine model [2, 11]. State machine-based specification models a software system with a number of states that the software system can achieve, and the transitions among these states. Each feasible path of transitions [2] derived from a state machine model represents an operational scenario of the software system. Therefore the instances of the operational scenarios will form a set of test scenarios for software testing. However, since cycles in the state machine model may lead to infinite number of feasible paths of transitions, exhaustive testing is deem impossible. One important issue is to determine which feasible paths should be selected for software testing. A default criterion for designing test cases with reference to state machine model is that all transitions in the model are covered by the test executions. This is called the all-transitions coverage criterion [11] which means each transition specified in the state machine model is triggered

IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 4, Issue 2, April - May, 2016
ISSN: 2320 – 8791 (Impact Factor: 2.317)
www.ijreat.org

at least once by executing the test cases. In [6], a formal mechanism has been developed, which is theoretically based on FCA, for selecting a reduced set of test scenarios that can satisfy the all-transitions coverage criterion.

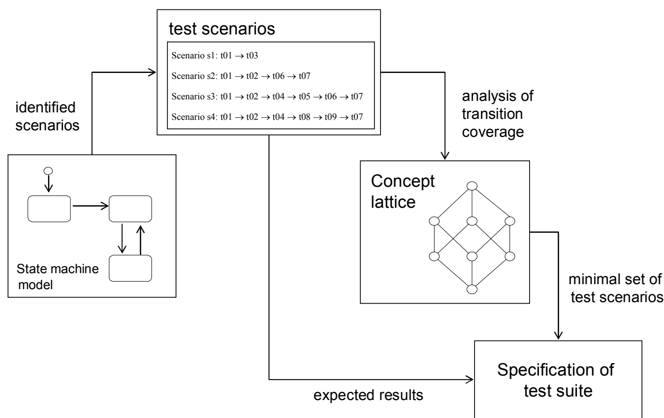

Fig. 2 Selecting test scenarios with FCA

Figure 2 summarizes the process of the mechanism. First, we start with a state machine model of a designated system. By traversing the state machine model, we can discover a set of possible test scenarios. Next, we apply FCA to analyze the formal context of the test coverage relationship between the set of test scenarios and the set of transitions specified in the state machine model. The outcome is a minimal set of test scenarios for software testing.

The methodology of our approach [7] involves five steps together with a set of pre-conditions and post-conditions which are related to the applicability of our approach in real situation.

Pre-conditions:

- The software functional requirements are expressed in form of state machine based specification;
- The feasible transition paths specified in the state machine based specification can be executed with the state changes in the target system observable.

Post-conditions:

- The derived test scenarios and the expected results are expressed with the terminology used in the user's context;
- Verdicts of the testing results can be obtained by comparing the running results with the expected results.

Steps:

Step 1: Deriving test scenarios: given a state machine based specification, by traversing the state machine model, a set of feasible transition paths can be derived as the test scenarios for software testing purposes.

Step 2: Specifying transition coverage: in the context of transition coverage, FCA is 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.

Step 3: Building concept lattice: a set of concepts can be formed by analyzing the transition coverage of the test scenarios. The concepts will be organized hierarchically to form a concept lattice.

Step 4: Determining minimal set of test scenarios: by utilizing the properties of concept lattice, we can incrementally determine a minimal set of test scenarios with adequate test coverage.

Step 5: Specifying the test suite: the selected minimal set of test scenarios, together with the expected running results, are specified to form a specification of the test suite for testing the target system.

Furthermore, during whole software development life cycle, changes and maintenance of software requirements needed to be carefully handled. When software requirements change, the corresponding test scenarios for software testing will also evolve. Through incremental updating the concept lattice structure, our approach can also support incremental updates of the minimal test suite for evolving software requirements.

4. Related Work

In relation to software testing, Tallam and Gupta [9] also adopted FCA to present a Delayed-Greedy heuristic for selecting the minimum number of test cases for testing a given set of testing requirements. However, because of the involvement of attribute reduction procedure, their approach may not support incremental update of the test suite for the situations that when some new test cases have been derived from evolving software requirements.

Sampath et al. [8] have also applied FCA in test suite reduction for web applications testing. Their approach

considers each of the URLs used in a web session as a formal attribute and each web session as a formal object which constitutes to be a test case. The reduced test suite is derived by selecting those test cases associated with the strongest concepts (the concept nodes that are just above the bottom-most concept node in the concept lattice). Although the method can support incremental selection of test cases, the resultant test suite may not be minimal since redundancy may still exist among the strongest concepts. By utilizing the incremental mechanism for updating the concept lattice structure [7], our approach can iteratively identify any test scenarios which turn out to be redundant when new test scenarios are added. Those redundant test scenarios will be removed in order to maintain the test suite minimal.

Genetic algorithms are search techniques based on natural genetic and evolution mechanisms for solving optimization problems. Genetic algorithms typically start with a random population of solutions, called chromosomes. Then, the initial solution undergoes a series of recombination and mutation processes and evolves into a target solution. In relation to software testing, genetic algorithms have been applied in test data generation. In particular, Doungsa-ard et al. [3] applied genetic algorithms in generating test data from state machine model. The chromosome used in their method is a sequence of events that can trigger the transitions specified in the given state machine model. However, the coverage of transitions varies and depends on the length of chromosome, and thus, their method cannot always achieve full coverage of transitions. Our research work, by analyzing the properties of the concept lattice structure, can be used for checking the adequacy of test coverage [7] so as to ensure that the selected test scenarios can satisfy the all-transition coverage criterion.

5. Conclusion

FCA provides a mathematical foundation for combining and organizing individual concepts of a given context to form a concept lattice. This paper summarized the experience of using FCA in supporting software testing. The FCA mechanism is particularly useful in supporting model-based software testing.

Our approach makes use of the concept analysis mechanism to support incremental reduction of model-based test suite [7] with reference to state machine model, which is used for modeling the functional requirements of software systems. By executing a set of model-based test scenarios, the conformance of the target system to its requirements can be tested. In analyzing the test coverage, FCA works as a sound mathematical foundation for analyzing the association between the model-based test scenarios and the coverage requirements for determining a minimal set of test suite.

The two major advantages of our approach are:

(1) Guidance of test case design: with reference to the state machine based specification, test scenarios can be derived based on the feasible transition paths. This could help the development team in designing the test cases and preparing the test review.

(2) Cost saving: research studies show software testing is an extremely costly and time consuming process [5]. Our approach is able to determine a minimal set of test scenarios whilst maintaining adequate test coverage. This could save the cost in test cases execution, and thus, save the cost of software development.

References

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

[2] L.C. Briand, Y. Labiche, , and J. Cui, “Automated support for deriving test requirements from UML statecharts”, Software and Systems Modeling, vol. 4, no. 4, 2005, pp.399–423.

[3] C. Doungsa-ard, K. Dahal, A. Hossain, T. Suwannasart, “Test Data Generation from UML State Machine Diagrams using GAs”, Proceedings of International Conference on Software Engineering Advances, ICSEA 2007, pp. 47–53.

[4] B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, 1999.

[5] M.J. Harrold, “Testing: a roadmap”, ICSE - The Future of Software Engineering Track, Limerick, Ireland, 4–11 June 2000, pp.61–72.

[6] P. Ng and R.Y.K. Fung, “A Concept Lattice Approach for Model-based Test Suite Reduction”, Journal of Electronics and Computer Science, vol.10, no. 2, ISSN 1229-425X, 2008, pp.105-112.

[7] P. Ng, R.Y.K. Fung, and R.W.M. Kong, “Incremental Model-based Test Suite Reduction with Formal Concept Analysis”, Journal of Information Processing System, vol.6, no. 2, ISSN 1976-913X, 2010, pp.197-208.

[8] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A.S. Greenwald, “Applying Concept Analysis to User-Session-

Based Testing of Web Applications", IEEE Transactions on Software Engineering, vol.33, no.10, 2007, pp.643–658.

[9] 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, vol.31 no.1, pp.35– 42.

[10] T. Tilley, R. Cole, P. Becker, and P. Eklund P, "A survey of formal concept analysis support for software engineering activities", Formal Concept Analysis, LNAI 3626, Ganter et al. (eds.), Springer-Verlag Berlin Heidelberg, 2005, pp. 250–271.

[11] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.