Deriving Test Cases from B Machines Using Class Vectors

W. L. Yeung[1] and K. R. P. H. Leung[2]

[1] Lingnan University, Hong Kong

[2] Hong Kong Institute of Vocational Education, Hong Kong

Abstract. This paper proposes a specification-based testing method for use in conjunction with the B method. The method aims to derive a set of legitimate class vectors from a B machine specification and it takes into account the structure and semantics of the latter. A procedure for test case generation is given. One advantage of the method is its potential to be integrated with the B method via its support tools.

1 Introduction

Formal methods play an important role in the verification and validation of enterprise information systems. Formal methods emphasize the formulation of a precise formal specification and the formal verification of a design against its specification. From the point of view of software testing, a specification serves as a basis for functional (black-box) testing on which test cases are systematically derived. Recently, there have been much interests in methods and tools for generating test cases from formal specifications (see, e.g. [2]). While some of these approaches aim to fully automate the testing process (e.g. [5]), others attempt to bring the existing body of knowledge in software testing to formal-specification-based testing (e.g. [4]).

This paper discusses a method based on earlier work on software testing using class vectors [6] for generating test cases from formal specifications written in the B Abstract Machine Notation (AMN) [1]. The class vectors method was developed to overcome some shortcomings of the Classification Tree method [3]. It defines classes and classifications in a formal setting and defines conditions formally for generating test cases. It is also a general testing method applicable to software with only informal specifications. The main contribution of this paper is to adopt and adapt the class vectors method for the B AMN formal specification language, which is notational basis of the B method. The B method is a major formal method for the development of information systems.

The rest of this paper is organized as follows. The next section first describes a B AMN specification for a small information system example. Section 3 defines the problem of generating a set of legitimate test cases based on the concept of classes. Section 4 and 5 define two necessary conditions for partitioning and filtering class vectors. Section 6 presents the test case generation procedure and illustrate it with the example. Section 7 discusses some possible extensions of the method and Section 8 gives a conclusion.

L. Yeung W. and R. P. H. Leung K. (2005).
Deriving Test Cases from B Machines Using Class Vectors.
In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 71-76
DOI: 10.5220/0002571500710076

MACHINE *CreditCard*$(itype, iclass, ischeme)$
SETS $CARDCLASS = \{platinum, gold, classic\}$;
$CARDTYPE = \{visa, mastercard\}$;
$REWARDSCHEME = \{airticket, gift\}$;
$AIRLINE = \{cx, dg\}$;
$TICKETTYPE = \{first, business, economy\}$;
$SUPPLIER = \{reda, tdc\}$;
PROPERTIES
$itype \in CARDTYPE \wedge iclass \in CARDCLASS \wedge ischeme \in REWARDSCHEME$
VARIABLES $type, class, scheme, balance$
INVARIANT
$type \in CARDTYPE \wedge class \in CARDCLASS \wedge scheme \in REWARDSCHEME \wedge$
$(class = platinum \Rightarrow balance \in 0..8000) \wedge$
$(class = gold \Rightarrow balance \in 0..4000) \wedge$
$(class = classic \Rightarrow balance \in 0..1000)$
INITIALISATION
$type := itype \parallel class := iclass \parallel scheme := ischeme \parallel balance := 0$
OPERATIONS
updatebalance$(b) \mathrel{\widehat{=}}$
PRE $(class = platinum \Rightarrow b \in 0..8000) \wedge (class = gold \Rightarrow b \in 0..4000) \wedge$
$(class = classic \Rightarrow b \in 0..1000)$
THEN $balance := b$ **END;**
redeemair$(al, tt, \ldots) \mathrel{\widehat{=}}$
PRE $scheme = airticket \wedge al \in AIRLINE \wedge$
$(al = cx \Rightarrow tt \in TICKETTYPE) \wedge (al = dg \Rightarrow tt \in \{first, business\}) \ldots$
THEN ... **END;**
redeemgift$(su, \ldots) \mathrel{\widehat{=}}$
PRE $scheme = gift \wedge su \in SUPPLIER \wedge \ldots$
THEN ... **END;**
...
END

Fig. 1. A formal specification of a credit card object in B AMN

2 An Example

We use a small information system example about credit card data processing adapted from [6]. The ABC Bank issues both Visa and MasterCard credit cards in three different classes, namely platinum, gold, and classic. Credit limits for these three different classes are $8000, $4000, and $1000, respectively.

The ABC Bank operates two reward schemes for its credit card customers. Every credit card holder of ABC Bank can join only one of these two schemes and earn reward points for using the credit cards for purchases. For scheme A, reward points earned can only be used to redeem free air tickets from airlines CX or DG. Redeemed air tickets can be of first, business, or economy class with airline CX, and of first or business class tickets with airline DG. For scheme B, reward points earned can be used to redeem free gifts from suppliers RedA or TDC.

Figure 1 shows a (partial) formal specification of a credit card object in the B Abstract Machine Notation. For readers who are not familiar with B AMN, a B specification is structured as a number of B machines (which roughly correspond to objects), each of which may have its own state and operations on its state. A B machine may include or reference other B machines to become a more complex machine and that is how complex specifications are structured. Each machine specification consists of a number of optional clauses, each beginning with a keyword in capital. In Figure 1, the MACHINE clause gives the B machine a name (*CreditCard*) and defines three parameters of the machine. The SETS clause define some sets of values used in the machine. The PROPERTIES clause define the types of the machine parameters. The state variables of the machine are defined in the VARIABLES clause and we can provide invariant properties for these variables in the INVARIANT clause. The INITIALISATION clause defines operations for initializing the state variables, whereas the OPERATIONS clause lists the operations available for updating and reading the state variables. An operation may take input parameters and has a precondition (preceded by the keyword PRE) on which the operation is guaranteed to keep the machine in a consistent state.

In order to generate (functional) test cases for the *CreditCard* B machine, we need to first identify the input variables and their legitimate values. The three input variables are on the first line ($itype, iclass, ischeme$) are actually parameters of the B machine; they provide initial values to the state variables ($type, class, scheme$). Their legitimate values are defined in the **PROPERTIES** section. Other input variables come from the parameters of the operations and their legitimate values are defined by the preconditions:

- variable *b*'s legitimate values depend on the class of the credit card
- variable *al* must be either *cx* or *dg*; variable *tt* depends on *al*
- variable *su* must be either *reda* or *tdc*

3 The Problem

Let X be an input variable and $\tau(X)$ be the set of values that X can take, i.e. the type of X . Assume that $\tau(X)$ can be partitioned into a finite number k of disjoint subsets, $\tau^1(X), \ldots, \tau^k(X)$ and we call them the classes of X. Let $class_X$ be a function that takes an input value of X and returns the class that it belongs, i.e. $class_X(X) = \tau^k(X)$ if and only if $X \in \tau^k(X)$.

Given a vector of n input values, $(X_1, \ldots, X_n)$, denoted by $\bar{X}$, the following function is defined:

$$classes(\bar{X}) = (class_{X_1}(X_1), \ldots, class_{X_n}(X_n))$$

and we call this a class vector.

Given a B machine with n input variables $X_1, \ldots, X_n$ and for each input variable X_i, m_i different classes $\tau^1(X_i), \ldots, \tau^{m_i}(X_i)$, there are $m_1 \times \ldots \times m_n$ possible different class vectors.

However, not all class vectors represent legitimate inputs. The predicates defined in a B machine specification help us to constrain the set of class vectors to only those

that provide legitimate inputs. For instance, a credit card object initialized as *classic* cannot legitimately accept 8000 as an input value of variable *b* of the operation **updatebalance**—the precondition of **updatebalance** rules out any classes of values except $0..1000$ for *b* in the case of *classic*.

The problem is how to define such a set of legitimate class vectors based on the B machine specification and then how to generate test cases from the set.

4 Independent Input Variables

To define the set of legitimate class vectors for a B machine specification, we first divide the set of input variables into independent sets of input variables.

Definition 1 (Independence) *Two input variables X_1 and X_2 are independent to each other if and only if the B specification admits any combination of values of them as input, i.e.*

$$\forall x_1 \in \tau(X_1), x_2 \in \tau(X_2) \bullet [T]I \wedge \bigwedge_i [T](I \wedge Pre_i)$$

where T is the initialisation clause of the B machine, I is the invariant, and Pre_i is the precondition of the i operation.

Consider the input variables *itype*, *iclass*, and *b* of the *CreditCard* machine. *itype* is clearly independent from the other two as it does not relate to the other two variables (or any other variables) in the Invariant or any Preconditions. The other two variables, *iclass* and *b*, relate to each other in the first precondition (via the initialisation clause) and are therefore not independent.

For the seven input variables of the *CreditCard* machine, we identify the following independent sets of input variables:

$$\{itype\}, \{iclass, b\}, \{ischeme, al, tt, su\}$$

5 Coexisting Classes

After identifying the independent sets of input variables, we identify those classes of values that can coexist in a legitimate input class vector.

Definition 2 (Coexitence) *Given two input variables X_1 and X_2, and two respective classes of their values $\tau^i(X_1)$ and $\tau^j(X_2)$, these two classes are coexisting if and only if the B specification admits any combination of values of them in the input, i.e.*

$$\forall x_1 \in \tau^i(X_1), x_2 \in \tau^j(X_2) \bullet [T]I \wedge \bigwedge_i [T](I \wedge Pre_i)$$

where T is the initialisation clause of the B machine, I is the invariant, and Pre_i is the precondition of the i operation.

Consider the input variables *at* and *tt*. Variable *at* has the following classes $\{cx\}$ and $\{dg\}$. We can divide the values of *tt* into the following classes $\{first, business\}$ and $\{economy\}$ and call them $C1$ and $C2$. Referring to the precondition of the second operation of the *CreditCard* machine, it is clear that class $\{cx\}$ is coexisting with both $C1$ and $C2$ whereas class $\{dg\}$ is only coexisting with $C1$.

6 Test Case Generation

Based on the concepts described in the above sections, the generation of test cases from a B specification based on class vectors encompasses the following steps:

1. Identification of classes
2. Identification of independent sets of input variables
3. For each independent set of input variables, enumerate all class vectors containing only coexisting classes
4. Derive the Cartesian product of the independent sets of coexisting class vectors
5. For each class vector of the Cartesian product, select a vector of representative values as a single test case

For the Credit Card example, we have already identified the independent sets of input variables in section 4. The coexisting class vectors for these sets are as follows:

$$\begin{aligned}
\{itype\} &: (visa), (mastercard) \\
\{iclass, b\} &: (platinum, 0..1000), (platinum, 1001..4000), \\
&\quad (platinum, 4001..8000), (gold, 0..1000), \\
&\quad (gold, 1001..4000), (class, 0..1000) \\
\{ischeme, al, tt, su\} &: (airticket, cx, \{first, business\}), (airticket, cx, \{economy\}), \\
&\quad (airticket, dg, \{first, business\}), (gift, SUPPLIER)
\end{aligned}$$

Taking the Cartesian product of the above sets yields $2 \times 6 \times 4 = 48$ different class vectors, from which we can derive 48 legitimate test cases.

7 Discussion

The concept of a class (of input values) goes back to the classification tree method for black box testing. In the test case generation method presented in this paper, we define classes as disjoint sets of input values of single variables and do away with the concept of classification as our starting point is already a formal specification with well defined input variables, rather than an informal specification. As a result, the concept of a class vector is slightly simplified as compared to the one defined in [6].

On the other hand, the derivation of legitimate class vectors still relies on the independence and coexistence conditions which are now more rigorously defined in terms of the B Abstract Machine Notation (AMN) semantics. Note that for the lack of space and the sake of clarity the definitions cover only the invariant part and operation preconditions of the B specification and they can be extended to cover other specification clauses such as CONSTRAINTS and ASSERTIONS.

For similar reasons, the applicability of the proposed method for large specifications involving tens or hundreds of B machines has not been illustrated in this paper. This would actually be taken care of by the semantics of the modular constructs of B AMN. For instance, in a composite B machine with M_1 including M_2 and M_3, the invariant and operation preconditions of M_1 has to take into account those of M_2 and M_3 for any M_1 operations that affect the states of M_2 and M_3. It suffice to say that for large specifications with complex inter-relationships among many B machines, the use of support tools to help keep track of all the relationships is essential. In fact, the test case generation method presented in this paper should ideally be integrated in support tools such as the B-toolkit for the B method. The independence and coexistence conditions would be generated in a similar way as proof obligations that can be verified with the help of proof assistance and proof checker.

8 Conclusion

We adopted and adapted a previous black-box testing method based on class vectors for use in the formal setting of the B AMN specification language. The formal definition of class vectors and the conditions for deriving legitimate test cases from B machines are defined. We also outline the procedure for test case generation and illustrate it with an example in this paper. The main advantage of this approach is that it takes into account the structure and semantics of the B AMN specification language and can be readily integrated with the B method via its support tools. This is also the aim of our further work.

References

1. J. R. Abrial. *The B-Book*. Cambridge University Press, 1996.
2. M. C. Gaudel. Testing can be formal too. In *TAPSOFT'95*, pages 82–96. Springer, 1995.
3. M. Grochtmann and K. Grimm. Classification Trees for Partition Testing. *Software Testing, Verification and Reliability*, 3:63–82, 1993.
4. Robert M. Hierons, Mark Harman, and Harbhajan Singh. Automatically Generating Information from a Z Specification to Support the Classification Tree Method. In *ZB 2003*, volume 2651, pages 388–407. Springer, 2003.
5. Bruno Legeard, Fabien Peureux, and Mark Utting. A Comparison of the BTT and TTF Test-Generation Methods. In *ZB 2002*, volume 2272, pages 309–329. Springer, 2002.
6. Karl R. P. H. Leung and Wai Wong. Deriving Test Cases Using Class Vectors. In *Proc. 7th Asia-Pacific Software Engineering Conference*, pages 146–153. IEEE, 2000.