2008 International Conference on Computer Science and Software Engineering

A Scheme to Aid Construction of Left-Hand Sides of Axioms in Algebraic Specifications for Object-Oriented Program Testing

Huo Yan Chen, Lin Tan
Department of Computer Science
Jinan University
Guangzhou, 510632, P.R. China
tchy@jnu.edu.cn

T.H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

***Abstract*—In order to ensure reliability and quality, software systems must be tested. Testing object-oriented software is harder than testing procedure-oriented software. It involves four levels, namely the algorithmic level, class level, cluster level, and system level. We proposed a methodology TACCLE for class- and cluster- level testing. It includes an important algorithm GFT for generating fundamental equivalent pairs as class-level test cases based on axioms in a given algebraic specification for a given class. This formal methodology has many benefits. However, system analysts often find it difficult to construct axioms for algebraic specifications. In this paper, we propose a scheme to aid the construction of the left-hand sides of axioms. The scheme alleviates the difficulties of the system analysts and also helps them check the completeness, consistency, and independence of the axiom system.**

Keywords—testing; object-oriented; algebraic specification; axiom; prototype tool

I. INTRODUCTION

Object-oriented program, despite its popularity, it also poses challenges to software testers. The testing of object-oriented programs involves four levels, namely the algorithmic level, class level, cluster level, and system level [1]. Testing at the algorithmic and system levels is similar to that for traditional programs. However, testing at the class and cluster levels require new techniques. In [2, 3, 4], we proposed a methodology TACCLE for object-oriented class- and cluster-level testing. Class-level testing is more basic. It includes generating test cases, executing test cases, and determining whether the results of execution of test cases conform to requirements.

Given an algebraic specification of a class under test, we define a *fundamental pair* as two equivalent ground terms generated by replacing all the variables on both sides of an axiom in the specification with normal forms. We proposed an algorithm known as *GFT*, for *G*enerating *F*inite number of fundamental pairs as *T*est cases [3]. Each fundamental pair corresponds to a pair of equivalent method sequences in an implementation. If two objects resulting from the executions of two method sequences corresponding to a fundamental pair are not observational equivalent, then a failure in the implementation of the class is found and reported. GFT is based on the axioms in a given algebraic specification for the class under test. It has many benefits.

Practicing system analysts and testers often find it difficult to construct equational axioms for formal specifications. If a company uses semi-formal specifications, such as timing diagrams in the case of ASM Assembly Automation Ltd. [5], we can develop an automatic tool to transform the graphic specifications into algebraic specifications, keeping the latter internal to the testing tool and transparent to the user. In this technology-transfer example, the difficulty is relaxed by constructing the algebraic axioms from the timing diagrams via the concept of communicating finite-state machines. Paper [5] reports the details of this real-life experience. On the other hand, if a company uses informal specifications, in order to apply TACCLE to testing, analysts or testers need to manually construct axioms for the algebraic specifications from the informal counterpart.

This paper proposes a scheme to help analysts and testers to construct the left-hand sides of axioms in algebraic specifications for object-oriented program testing. The scheme alleviates the difficulties faced by analysts and also helps them check the completeness, consistency, and independence of the constructed axiom system.

The remainder of this paper is organized as follows: In Section 2, related concepts are summarized. A proposition, which is useful for the scheme, is presented in Section 3. The detail of the scheme to help analysts or testers construct the left-hand sides of axioms is presented in Section 4. The implementation and experiments of a tool to aid the scheme is presented in Section 5. Finally, some discussions and the conclusion are given in Section 6.

This research is supported by a Union Grant of Guangdong Province and National Natural Science Foundation of China (#U0775001), by a grant of the Guangdong Province Science Foundation (#7010116), and by a grant of the Youth Science Foundation of Jinan University (#51208035). Lin Tan is a master degree student supervised by Huo Yan Chen.

978-0-7695-3336-0/08 $25.00 © 2008 IEEE
DOI 10.1109/CSSE.2008.600

II. The Concepts

Related basic concepts have been explained in detail in [2, 3]. They include *algebraic specifications*, *axioms*, *terms*, *rewriting*, *normal forms*, *ground terms*, *canonical specifications*, *normally equivalent*, *creators*, *constructor*, *transformers*, *observers*, and *fundamental pairs of equivalent terms* (or simply *fundamental pairs*).

The following concepts and strategies are new.

Definition 1. Suppose $a_1, a_2, \ldots, a_k$ are axioms in an algebraic specification of a given class. Consider a new axiom a_{k+1}: $t_2 = t_1$. There are three cases:

(i) Term t_2 cannot be rewritten by any axioms in $\{a_1, a_2, \ldots, a_k\}$ (under the appropriate condition, if any);

(ii) Term t_2 can be rewritten to the normal form of term t_1 by some axioms in $\{a_1, a_2, \ldots, a_k\}$ (under the appropriate condition, if any). In this case, we say that a_{k+1} is *dependent on* $\{a_1, a_2, \ldots, a_k\}$ or a_{k+1} is not *independent of* $\{a_1, a_2, \ldots, a_k\}$.

(iii) Term t_2 can be rewritten by some axioms in $\{a_1, a_2, \ldots, a_k\}$ (under the appropriate condition, if any), but the resulting normal form is not the same as the normal form of term t_1 by using axioms in $\{a_1, a_2, \ldots, a_k\}$. In this case, we say that a_{k+1} *contradicts with* $\{a_1, a_2, \ldots, a_k\}$ or that a_{k+1} is *inconsistent with* $\{a_1, a_2, \ldots, a_k\}$.

Obviously, If a_{k+1} is *dependent on* $\{a_1, a_2, \ldots, a_k\}$, then adding a_{k+1} to $\{a_1, a_2, \ldots, a_k\}$ does not affect the normal form of any term. In other words, it is redundant with respect to $\{a_1, a_2, \ldots, a_k\}$. Thus, we have

Strategy 1. If a_{k+1} is *dependent on* $\{a_1, a_2, \ldots, a_k\}$, then we need not add a_{k+1} to $\{a_1, a_2, \ldots, a_k\}$ when constructing an algebraic specification.

On the other hand, it is also obvious that if a_{k+1} *contradicts with* $\{a_1, a_2, \ldots, a_k\}$ for a canonical specification, then after adding a_{k+1} to $\{a_1, a_2, \ldots, a_k\}$, the specification will no longer be canonical. Hence, we have

Strategy 2. For a canonical specification containing axioms $a_1, a_2, \ldots, a_k$, if a_{k+1} *contradicts with* $\{a_1, a_2, \ldots, a_k\}$, then we should not add a_{k+1} to $\{a_1, a_2, \ldots, a_k\}$ when constructing the canonical specification.

Definition 2. In case (i) of Definition 1, the new axiom a_{k+1} is not *dependent on* $\{a_1, a_2, \ldots, a_k\}$ and does not *contradict with* $\{a_1, a_2, \ldots, a_k\}$. In this case, we say that a_{k+1} is *independent of and consistent with* $\{a_1, a_2, \ldots, a_k\}$.

Definition 3. If every $a_i \in \{a_1, a_2, \ldots, a_k, a_{k+1}\}$ is *independent of and consistent with* $\{a_1, a_2, \ldots, a_k, a_{k+1}\} \setminus \{a_i\}$, then we say that $\{a_1, a_2, \ldots, a_k, a_{k+1}\}$ has *internal independence and consistency*.

III. The Proposition

The following proposition is useful for setting up a scheme to help construct the left-hand sides of axioms in algebraic specifications for object-oriented program testing via TACCLE.

Proposition 1. Suppose t_0 is a sub-term of term t. If t_0 appears as the left-hand side of an axiom (under an appropriate condition, if any) in an algebraic specification of a given class C, then t cannot appear as the left-hand side of another axiom (under the same condition, if any) in the algebraic specification of C.

Proof: Let $t = t_0.t_1$. Suppose there is an axiom a_i: $t_0 = s_0$ and another axiom a_k: $t = s$. We have

$$t = t_0.t_1 \overset{a_i}{\Rightarrow} s_0.t_1$$

If the term $s_0.t_1$ is equivalent to the term s, then the axiom a_k is dependent on $\{\ldots, a_i, \ldots\}$, so that a_k is redundant and should be deleted. Otherwise, the axiom a_k contradicts with $\{\ldots, a_i, \ldots\}$ and hence a_k must also be deleted.

IV. The Scheme

The *CLA* scheme to aid the *C*onstruction of *L*eft-hand sides of *A*xioms in algebraic specifications for object-oriented program testing via TACCLE consists of the following steps:

(1) By interacting with the requirement analyst, for a given class, determine and input the set *CR* of creators, the set *CT* of constructors or transformers, and the set *OB* of observers. They may contain parameters as appropriate.

(2) Let *PL* denote the set of *preliminary left-hand sides of* axioms, and let *PA* denote the set of *preliminary axioms* (*or* "*pre-axioms*" for short). Set $PL = \emptyset$; set $PA = \emptyset$.

(3) For each $cr \in CR$ do {

  For each $ob \in OB$ do {

    Ask analyst to give a term (including condition) as the right-hand side of a *pre-axiom ax* with *cr.ob* as the left-hand side [1], and set $PA = PA \cup \{ax\}$;

    If the condition of *ax* is not "always *true*", then iterate the previous step to constructing multiple axioms with the same left-hand side as *ax* but with mutually exclusive conditions;

  }

}

(4) For each $cr \in CR$ do {

  For each $ct \in CT$ do {

    Ask analyst whether *cr.ct* is made the left-hand side of a *pre-axiom*; [2]

    If yes, {

      Ask analyst to give the right-hand side (including condition) to construct a *pre-axiom ax*;

[1] Such as *new.empty* in axiom a_1 of Example 1 in [3]. Some attributes, say *empty and top*, may have related semantics. Thus, we need to check the consistency of their corresponding preliminary axioms a_1: *new.empty* = *true* and a_5: *new.top* = *NIL*. However, such kinds of consistency cannot be checked by axiom rewriting, and hence we leave the checking to step (9).

[2] Such as *new.pop* in axiom a_3 of Example 1 in [3].

If the condition of ax is not "always *true*", then iterate the previous step to constructing multiple axioms with the same left-hand side as ax but with mutually exclusive conditions;

Check whether ax is consistent with and independent of the set PA of *pre-axioms* constructed;

If yes, {

$PA = PA \cup \{ax\}$; [3]

For every previous *pre-axiom* ax_0 that can be derived from ax and others, set $PA = PA \setminus \{ax_0\}$

}

Else, {

Ask analyst to determine whether to submit another term as the right-hand side;

If submitting another one, then construct another *pre-axiom* ax and repeat the above process;

If no more submission, then $PL = PL \cup \{cr.ct\}$;

}

},

Else, skip it and set $PL = PL \cup \{cr.ct\}$;

};

};

(5) Take a variable A of object in the given class, and set $PL = PL \cup \{A\}$ and $PL0 = \varnothing$; [4]

(6) For each $X \in PL$ do {

For each $ct \in CT$ do {

Set $X = X.ct$;

Ask analyst whether X is made the left-hand side of a *pre-axiom*; [5]

If yes, {

Ask for the right-hand side (including condition) to construct a *pre-axiom* ax;

If the condition of ax is not "always *true*", then iterate the previous step to constructing multiple axioms with the same left-hand side as ax but with mutually exclusive conditions;

Check whether ax is consistent with and independent of the set PA of *pre-axioms* constructed;

If yes, {

$PA = PA \cup \{ax\}$; [6]

For every previous *pre-axiom* ax_0 that can be derived from ax and others, set $PA = PA \setminus \{ax_0\}$

}

Else, {

Ask analyst to determine whether to submit another term as the right-hand side;

If submitting another one, then construct another *pre-axiom* ax and repeat the above process;

If no more submission, then $PL = PL \cup \{cr.ct\}$;

}

};

If not or if the disjunction of conditions of all axioms (with the same left-hand side as that of ax) is not "always *true*", {

For each $ob \in OB$ do {

Ask analyst whether $X.ob$ is made the left-hand side of a *pre-axiom*; [7]

If yes, {

Remind analyst that the condition of the new *pre-axiom* must be mutually exclusive with that of the previous *pre-axioms* with X as the left-hand side;

Ask for the right-hand side (including condition) to construct a *pre-axiom* ax;

If the condition of ax is not "always *true*", then iterate the previous step to constructing multiple axioms with the same left-hand side as ax but with mutually exclusive conditions;

Check whether ax is consistent with and independent of the set PA of *pre-axioms* constructed;

If yes, {

$PA = PA \cup \{ax\}$;

For every previous *pre-axiom* ax_0 that can be derived from ax and others, set $PA = PA \setminus \{ax_0\}$

}

Else, {

[3] According to Proposition 1, we do not set $PL = PL \cup \{cr.ct\}$ here. This is an example of pruning.

[4] PL is for this loop, and $PL0$ is for the next loop.

[5] Such as *S.push(N).pop* in a_4 of Example 1 in [3], and *A.debit(N)* in a_{13} of Example 2 in [6].

[6] According to Proposition 1, we do not set $PL = PL \cup \{cr.ct\}$ here. This is another example of pruning.

[7] Such as *S.push(N).empty* in axiom a_2 of Example 1 in [3].

Ask analyst to determine whether to submit another term as the right-hand side;

If submitting another one, then construct another *pre-axiom ax* and repeat the above process;

// Note here that we do not set $PL0 = PL0 \cup \{X.ob\}$ when there is no more submission;

}

};

Else, skip it;

};

Set $PL0 = PL0 \cup \{X\}$;

};

};

};

(7) If $PL0 = \varnothing$, go to (9);

(8) Ask analyst whether it is the end of the construction of *pre-axioms*;

If not, $\{PL = PL0;\ PL0 = \varnothing;$ go to (6);};

(9) By interacting with analyst, convert the set *PA* of *pre-axioms* to the set *RA* of *required axioms* by selecting, checking, uniting, refining, changing the names of parameter variables, adding, or deleting, and so on. For axioms with left-hand sides that match one another, remind analyst that the conditions must be mutually exclusive.

(10) Output the set *RA* of the *required axioms* of the given class *C*; End the scheme.

V. Implementation and Experiments

We use Visual C++ 2005 under Microsoft Windows XP to implement a semi-automatic prototype tool for the CLA scheme. The tool uses dialog framework resources and control-widget resources to realize interaction with users.

Each axiom is represented by a structure *AxiomItem*, defined as follows:

```
struct AxiomItem {
    CString m_Left;  // left-hand side of the axiom
    CString m_Right;  // right-hand side of the axiom
    CString m_Co;  // condition of the axiom
    AxiomItem * next;  // pointer for linked list   };
```

The set of axioms in an algebraic specification is denoted by a linked list of *AxiomItem* structures.

Experiments have been conducted on various case studies, including, for example, a class *Book* in a library system and a class *SavAcct* of savings accounts in a bank system. The work has been implemented by the second author. Details are not given in this paper because of page limitation. Readers may refer to his master thesis [6] for more information.

VI. Discussions and Conclusion

A finite number of fundamental pairs can be generated as test cases by algorithm GFT in [3], which is based on the axioms of a given algebraic specification for a given class under test. This testing approach has many advantages. However, system analysts often find it difficult to construct axioms for algebraic specifications. This paper presents a scheme, named *CLA*, to help analysts or testers construct the left-hand sides of axioms in algebraic specifications. The scheme alleviates the difficulties faced by analysts and also helps them check the completeness, consistency, and independence of the constructed axiom system.

The *CLA* scheme is based on an analysis of the patterns of left-hand sides of axioms in algebraic specifications. It uses an enumeration technique with a pruning technique based on Proposition 1. The pruning technique reduces the number of loops in executing the scheme and enhances its efficiency greatly.

The implementation and experiments for a semi-automatic tool to aid the scheme is also described in this paper.

In general, for a given class, the numbers of creators, constructors and transformers, and observers are small, and the number of loops in the *CLA* scheme is not large when the pruning technique is employed. Thus, the CLA scheme should be effective in requirements engineering.

As future work, we would like to use Prolog to develop the semi-automatic tool. This will make it easier to implement the function to check whether a new *pre-axiom* is consistent with and independent of the set *PA* of *pre-axioms* constructed.

References

[1] M.D. Smith and D.J. Robson, "A framework for testing object-oriented programs," Journal of Object-Oriented Programming, vol. 5, no. 3, pp. 45–53, 1992.

[2] Huo Yan Chen, T.H. Tse, and T.Y. Chen, "TACCLE: a methodology for object-oriented software testing at the class and cluster levels," ACM Transactions on Software Engineering and Methodology, vol. 10, no. 1, pp. 56–109, 2001.

[3] Huo Yan Chen, T.H. Tse, F.T. Chan, and T.Y. Chen, "In black and white: an integrated approach to class-level testing of object-oriented programs," ACM Transactions on Software Engineering and Methodology, vol. 7, no. 3, pp. 250–295, 1998.

[4] Huo Yan Chen, T.H. Tse, and Y.T. Deng, "ROCS: an object-oriented class-level testing system based on the relevant observable contexts technique," Information and Software Technology, vol. 42, no. 10, pp. 677–686, 2000.

[5] T.H. Tse, F.C.M. Lau, W.K. Chan, P.C.K. Liu, and C.K.F. Luk, "Testing object-oriented industrial software without precise oracles or results," Communications of the ACM, vol. 50, no. 8, pp. 78-85, 2007.

[6] Lin Tan, The Design and Implementation of the Semi-automatic Aid Tool for Constructing Algebraic Specification, Master's Thesis, Department of Computer Science, Jinan University, Guangzhou, China, 2008.