A Formal Model of Conformity Testing of Inheritance for Object Oriented Constraint Programming

Khalid Benlhachmi and Mohammed Benattou

Laboratory of Research in Computer Science and Telecommunications
Faculty of Science, Ibn Tofail University Kenitra, Morocco

Abstract

This paper presents an approach for extending the constraint model defined for conformity testing of a given method of class to its overriding method in subclass using inheritance principle. The main objective of the proposed work is to find the relationship between the test model of an overriding method and its overridden method using the constraint propagation. Our approach shows that the test cases developed for testing an original method can be used for testing its overriding method in a subclass and then the number of test cases can be reduced considerably.

The implementation of this approach is based on a random generation of test data and analysis by formal proof.

Keywords: Conformity test, constraints resolution, formal specification, inheritance, valid data, test data generation

1. Introduction

The principle of testing is to apply input events to the Implementation under Test (IUT) and to compare the observed output events to the expected results. A set of input events and its corresponding results is generally called test case and can be generated from the IUT specification. The purpose of testing methods is to find the failures that are not detected during system normal operation and to define the relationship between the specifications and implementations of entities under test. Indeed, in object oriented modeling, a formal specification defines operations by collections of equivalence relations and is often used to constrain class and type, to define the constraints on the system states, to describe the pre- and post-conditions on operations and methods, and to give constraints of navigation in a class diagram. The object-oriented constraints (OOC) are specified by formal languages as OCL [1] and JMI [2], and are used for generating test data from formal specifications [3].

In this context, this paper introduces an optimizing approach to minimize the test sequences used for testing the conformity of an overriding method in object oriented models. The main idea of this work is the use of a technique that generates test data by exploiting the existing test sequences. Indeed, it is important to reuse the test result of overridden methods for testing the conformity of the overriding methods during inheritance operation. That is why our approach specifies all cases of conformity of an overriding method by using data extracted from conformity testing of the original methods.

The work presented in this paper allows to extend the constraint model defined in [4] for modeling the specification of an overriding method in subclass using inheritance principle. This work is based on our model of similarity concept [5, 6] for testing the conformity of overriding methods from conformity results of the original methods. The main objective is to reduce the number of test cases by using the test result developed in parent classes. We
present firstly, the relationship between the test model of overridden and overriding methods, and we show how to use the constraint model for extracting the possible cases of test values. Secondly, we show how it is possible to exploit the test result provided by the original class to test the methods in a sub class.

This paper is organized as follows: in Section 2 we present related works and similar approaches for generating test data from a formal specification, in Section 3 we describe theoretical aspects of our test process, and we define our test formal model of constraints, in Section 4 we present how the testing formal model can be used to generate test data during inheritance operation, and we show how the conformity testing of an overriding method can be deduced from its overridden method in parent class. Finally, in Section 5 we describe our approach with an example of conformity testing for an object oriented model.

2. Related Works

Most works have studied the problem of relating types and subtypes with behavioral specification in an object-oriented paradigm. These proposed works show how the contracts are inherited during method overriding and how the testing process can use the formal specification. In [4], we have presented the definition of a formal model of constraint, illustrating the relationship between pre-conditions, post-conditions and invariants of methods, and we have formalized a generic constraint of a given individual method of class that contains all constraints into a single logical predicate. The given model translates algebraically the contract between the user and the called method.

In [5], we have developed a basic model for the concept of methods similarity, the test is based only on a random generation of input data. In [6], we have generalized the basic model of similarity using analysis of partition and formal proof. In [7], the authors propose a randomly generation of test data from a JML specification of class objects. They classify the methods and constructors according to their signature (basic, extended constructors, mutator, and observer) and for each type of individual method of class, a generation of test data is proposed. In [8], the paper describes specially the features for specifying methods, related to inheritance specification; it shows how the specification of inheritance in JML forces behavioral sub-typing.

The work presented in [9] shows how to enforce contracts if components are manufactured from class and interface hierarchies in the context of Java. It also overcomes the problems related to adding behavioral contracts to Object-Oriented Languages, in particular, the contracts on overriding methods that are improperly synthesized from the original contracts of programmer in all of the existing contract monitoring systems. The work is based on the notion of behavioral sub-typing; it demonstrates how to integrate contracts properly, according to the notion of behavioral sub-typing into a contract monitoring tool for java. In [10], the authors treat the problem of types and subtypes with behavioral specifications in object-oriented world. They present a way of specification types that makes it convenient to define the subtype relation. They also define a new notion of the subtype relation based on the semantic properties of the subtype and super-type. In [11], they examine various notions of behavioral sub-typing proposed in the literature for objects in component-based systems, where reasoning about the events that a component can raise is important.

All the proposed works concerning the generation of test data from formal specification or to test the conformity of a given method implementation, use only the constraint propagation from super to subclass related to subtype principle and do not exploit the test results of the original method. As an example, several test oracles used in industry as JML compiler can generate the conformity test of an overriding method even if its original method in the parent
class does not conform to its specification. Our approach shows that this type of test is unnecessary and can be removed, and presents how we can reduce the test cases by using the test values developed for testing the original methods in a parent class.

3. Formal Model of Constraint

This section presents a formal model of the generalized constraint defined in [4] which provides a way for modeling the specification of an overriding method by inheritance from a super-class. Indeed, we establish a series of theoretical concepts in order to create a solid basis for testing the conformity of overriding methods in subclasses using the test result of the original methods.

3.1. Constraint Propagation in Inheritance

In [4] we have presented the definition of a formal model of constraint, illustrating the relationship between the pre-condition \( P \), the post-condition \( Q \) of a method \( m \) and the invariant \( Inv \) of the class \( C \): this constraint \( H \) is a logical property of the pair \( (x,o) \) (\( x \) is the vector of parameters \( x=(x_1,x_2,...,x_n) \) and \( o \) is the receiver object) such that:

\[
H(x,o) : P(x,o) \Rightarrow [Q(x,o) \land Inv(o)], \quad (x,o) \in E \times I_c
\]

Where \( I_c \) is the set of instances of the class \( C \) and \( E \) the set of input vectors of \( m \).

Indeed, the invocation of a method \( m \) is generally done by reference to an object \( o \) and consequently, \( m \) is identified by the couple \( (x,o) \). The logical implication in the proposed formula means that: each call of method with \( (x,o) \) satisfying the pre-condition \( P \) and the invariant before the call, \( (x,o) \) must necessarily satisfy the post-condition \( Q \) and the invariant \( Inv \) after the call. In the context of this work, we assume that the object which invokes the method under test is valid (satisfying the invariant of its class), thus, the objects used at the input of the method under test are generated from a valid constructor. This justifies the absence of predicate \( Inv \) of the object \( o \) before the call to \( m \) in the formula \( H \) (Figure 1).

![Figure 1. Specification of a Method m](Image)

The purpose of this paragraph is to establish a series of theoretical rules in order to evolve the constraint \( H \) of a method \( m \) of a super-class during the operation of inheritance. We consider a method \( m \) of a class \( C_2 \) which inherits from the class \( C_i \) such that \( m \) overrides a method of \( C_i \). The original method and its overriding method in the subclass \( C_2 \) will be denoted respectively by \( m^{(1)} \) and \( m^{(2)} \). \( (P^{(1)},Q^{(1)}) \) denote respectively the pre-condition, the post-condition of the method \( m^{(1)} \); and \( Inv^{(1)} \) the invariant of \( C_i \); and \( (P^{(2)},Q^{(2)}) \) denote respectively the specific pre-condition, post-condition of the overriding method \( m^{(2)} \), and \( Inv^{(2)} \) the specific invariant of the class \( C_2 \). \( (P^{(2)},Q^{(2)}) \) denote respectively the pre-condition, the post-condition of the method \( m^{(2)} \), and \( Inv^{(2)} \) the invariant of \( C_2 \) (Figure 2).

The results of this paragraph are based on the works of Liskov, Wing [12] and Meyer [13] who have studied the problem relating to types and subtypes with behavioral specification in an object-oriented (OO) paradigm. Indeed, a derived class obeys the Liskov Substitution Principle (LSP) if for each overriding method \( m^{(2)} \), the pre-condition \( P^{(2)} \) is weaker than the pre-condition \( P^{(1)} \) of the overridden method \( (P^{(1)} \Rightarrow P^{(2)}) \) , the post-condition \( Q^{(2)} \) is stronger
than the post-condition \(Q'\) of the overridden method \((Q' \Rightarrow Q)\), and the class invariant \(Inv'\) of the subclass \(C_2\) must be equal to or stronger than the class invariant \(Inv\) of the \(C_1\) \((Inv' \Rightarrow Inv)\).

As a result of LSP:

The pre-condition \(P'\) of \(m'\) is the disjunction of \(P\) and the specific pre-condition \(P'_2\) of \(m'\) (Figure 2): \(P' \Leftrightarrow (P \lor P'_2)\) (1)

The post-condition \(Q'\) of \(m'\) is the conjunction of the post-condition \(Q\) of \(m\) and the specific post-condition \(Q'_2\) of \(m'\) (Figure 2): \(Q' \Leftrightarrow (Q \land Q'_2)\) (2)

The invariant \(Inv'\) of the class \(C_2\) is the conjunction of the invariant \(Inv\) of the class \(C_1\) and the specific invariant \(Inv'\) of \(C_2\) (Figure 2): \(Inv' \Leftrightarrow (Inv \land Inv'\)) (3)

Based on the definition of the generalized constraint, we have:

\[
\begin{align*}
H^{(1)}&(x,o) : P^{(1)}(x,o) \Rightarrow \{Q^{(1)}(x,o) \land Inv^{(1)}(o)\}, (x,o) \in E \times C_1 \text{: The constraint of } m^{(1)}.
\end{align*}
\]

\[
\begin{align*}
H^{(2)}&(x,o) : P^{(2)}(x,o) \Rightarrow \{Q^{(2)}(x,o) \land Inv^{(2)}(o)\}, (x,o) \in E \times C_2 \text{: The constraint of } m^{(2)}.
\end{align*}
\]

Using (1),(2),(3) the constraint of \(m'\) will have the following form:

\[
\begin{align*}
H^{(3)} : [ P \lor P'_2 ] & \Rightarrow [ Q^{(1)} \land Inv^{(1)} \land Q'_2 \land Inv'_2 ]
\end{align*}
\]

![Figure 2. Constraints of \(m^{(1)}\) and \(m^{(2)}\)](image)

In our approach, the specification \((P^{(2)}, Q^{(2)}, Inv^{(2)})\) of \(m'\) is constituted by:

Two inputs: Basic Input \((BI=P^{(1)})\) and Specific Input \((SI=P'_2)\) of \(m^{(2)}\).

Two outputs: Basic Output \((BO=(Q^{(1)}, Inv^{(1)})\)) and Specific Output \((SO=(Q'_2, Inv'_2))\) of \(m'\).

This induces 4 possible combinations of I-O: \((BI,BO),(BI,SO),(SI,BO),(SI,SO)\).

### 3.2. Constraint Model of Overriding Methods

The aim of this paragraph is to construct a formal model of an overriding method by generalizing the constraint model of the original methods. In achieving this goal, we should define a specific constraint of inheritance and also study the compatibility between the overriding methods and original methods in the parent class.

In this sense, we had proposed in [5,6] the definition of similarity concept for assuring if the overriding method \(m^{(1)}\) has the same behavior as its original version \(m^{(1)}\) in the super-class relatively to their common specification \((P^{(1)}, Q^{(1)}, Inv^{(1)})\). Indeed, the implementation in the subclass must override the implementation in the super class by providing a method \(m^{(2)}\) that is similar to \(m^{(1)}\).

We propose firstly the definition of a constraint \(H_{(SI,SO)}\) allowing a partial view of the overriding method \(m^{(2)}\) in the class \(C_2\). Secondly, we determine the relationship between the constraints \(H^{(2)}, H^{(1)}\) and \(H_{(SI,SO)}\).
The constraint \( H_{(SI, SO)} \) specifies the logical relationship between the input specific predicate \( P_2' \) of \( m^{(2)} \) and the output predicates \((Q_2', Inv_2')\) specific to \( m^{(2)} \).

**Definition 1:** (Constraint \( H_{(SI, SO)} \))

We define the constraint \( H_{(SI, SO)} \) of an overriding method \( m^{(2)} \) of a sub-class \( C_2 \) as a logical property of the pair \((x, o)\) such that:

\[
H_{(SI, SO)}(x, o): P_2(x, o) \Rightarrow [Q_2(x, o) \land Inv_2'(o)]
\]

Where \( I_{C_2} \) is the set of instances of \( C_2 \), and \( E \) is the set of the parameters vector of \( m^{(2)} \).

The logical implication in the proposed formula means that: each call of the method \( m^{(2)} \) with \((x, o)\) satisfying the specific precondition \( P_2' \) before the call, \((x, o)\) must necessarily satisfy the specific post-condition \( Q_2' \) and the specific invariant \( Inv_2' \) after the call of \( m^{(2)} \).

In the same way, we put:

\[
H_{(BI, SO)}(x, o): P_1(x, o) \Rightarrow [Q_1(x, o) \land Inv_1'(o)]
\]

And \( H_{(SI, BO)}(x, o): P_2'(x, o) \Rightarrow [Q_2'(x, o) \land Inv_2'(o)] \)

The relationship between \( H^{(1)} \) and \( H^{(2)} \) for the two similar methods \( m^{(1)} \) and \( m^{(2)} \) in classes \( C_1 \) and \( C_2 \) is shown in the following theorem:

**Theorem 1:**

\[
[H^{(2)} \Leftrightarrow (H^{(1)} \land H_{(SI, SO)}) ] (R)
\]

**Proof:** Using the following result: \([a \lor b] \Rightarrow (c \land d) \Rightarrow ([a \Rightarrow c] \land (a \Rightarrow d) \land (b \Rightarrow c) \land (b \Rightarrow d) )\]

We have: \([((P^{(1)} \lor P_2') \Rightarrow ((Q^{(1)} \land Inv^{(1)}) \land (Q_2' \land Inv_2')))) \Rightarrow [((P^{(1)} \Rightarrow (Q^{(1)} \land Inv^{(1)}) \land (P_2' \Rightarrow (Q_2' \land Inv_2'))))] \Rightarrow\]

Therefore, we have the following result: \([H^{(2)} \Leftrightarrow (H^{(1)} \land H_{(BI, SO)} \land H_{(SI, BO)} \land H_{(SI, SO)})]\)

In principle, the method \( m^{(2)} \) is not intended to provide the services \((SI, BO)\) and \((BI, SO)\) however \( m^{(2)} \) must guarantee the services \((BI, BO)\) (Figure 3) and \((SI, SO)\) (Figure 4).

Consequently, we put: \( H_{(BI, SO)} = 1 \) and \( H_{(SI, BO)} = 1 \).

Finally, the relation \((R)\) is equivalent to \([H^{(2)} \Leftrightarrow (H^{(1)} \land H_{(SI, SO)})]\).

---

**Figure 3. Constraint BI-BO of an Overriding Method**

**Figure 4. Constraint SI-SO of an Overriding Method**
The constraints \( H(1), H_{(st, so)} \) and the relation \( (R) \) form the theoretical basis that will be used to test the conformity of overriding methods in subclasses from the test result of their original methods in the parent classes.

4. Conformity Testing

The formal model of test proposed in [4] defines the notion of method validity in a basic class and constitutes a way to generate test data for conformity. In a conformity test, the input data must satisfy the precondition of the method under test. In this context, we are particularly interested in valid input values (i.e., the pairs \((x, o)\) that satisfy the precondition \( P \)). The conformity test for a method means that if a pre-condition is satisfied at the input, the post-condition and the invariant must be satisfied at the output. The purpose of this section is to generalize the model of [4-6] in order to test the conformity of an overriding method \( m(2) \) in a derived class by using the elements of conformity test of its original method \( m(1) \) in the superclass.

In [4], we have tested the conformity of methods in a basic class without taking into account the inheritance relationship: the model of test generates random data at the input of a method using elements of the valid domain which satisfy the precondition of the method under test. This test stops when the constraint \( H \) becomes False \( (H(x, o)=0) \) or when the maximum threshold of the test is reached with \( H \) satisfied.

**Definition 2: (Valid method)**

A method \( m \) of class \( C \) is valid or conforms to its specification if for each \((x, o)\in E\times I \), the constraint \( H \) is satisfied: \( \forall (x, o)\in E\times I : H(x, o)=1 \)

In other words, for a valid method: \( \forall (x, o) \in E\times I : (x, o) \) satisfies the precondition \( P \) then this \((x, o)\) must satisfy the post-condition \( Q \) and the invariant \( Inv \).

The conformity test of \( m(2) \) requires the following steps:

1. **Step 1:** a conformity test of a basic constructor of class \( C_2 \). This step is necessary for using valid objects at the input of the method under test.

2. **Step 2:** a similarity test of \( m(1) \) and \( m(2) \) relatively to \( (P(1), Q(1), Inv(1)) \).

We assume that the test of step 2 showed that \( m(1) \) and \( m(2) \) are similar. The conformity test process of the overriding method \( m(2) \) relatively to \( H(2) \) is based on the test result of \( m(1) \) relatively to \( H(1) \) and the test result of \( m(2) \) relatively to \( H_{(st, so)} \) (theorem1).

The conformity test of \( m(1) \) induces two cases: \( m(1) \) conforms to its specification or \( m(1) \) is not in conformity with its specification:

- **Case 1:** The method \( m(1) \) is not in conformity with its specification

  We have the following result (Figure 5):

**Theorem 2:**

If the overridden method \( m(1) \) in parent class \( C_1 \) is not in conformity with its specification \( (P(1), Q(1), Inv(1)) \) then any similar method \( m(2) \) in a subclass is not in conformity with its specification \( (P(2), Q(2), Inv(2)) \).

**Proof:**

We assume that the method \( m(1) \) is not in conformity with its specification. This means (Def.2) that: \( \exists (x_0, o_0) \in E\times I_{C1} : H^{(1)}(x_0, o_0)=0 \)
The object \( o_0 \) is an instance of the class \( C_1 \). We consider an object \( o_0' \) of the subclass \( C_2 \) that has the same values as the object \( o_0 \) for attributes of \( C_1 \). And therefore, the object \( o_0' \) has the same behavior as \( o_0 \) in a context of \( C_1 \) and consequently: \( \exists(x_0,o_0') \in E \times I_{C_2} : H^{(1)}(x_0, o_0')=0 \)

We apply the relationship \((R)\) (theorem1) on the pair \((x_0, o_0')\) of the domain \( E \times I_{C_2} \):

\[(R) \iff [H^{(2)}(x_0, o_0') \iff (0 \land H_{(SI, SO)}(x_0, o_0'))] \iff [H^{(2)}(x_0, o_0') \iff 0] \]

This means: \( \exists(x_0, o_0') \in E \times I_{C_2} : H^{(2)}(x_0, o_0')=0 \)

This shows (Def.2) that the method \( m^{(2)} \) is not in conformity with its specification.

- **Case 2**: The method \( m^{(1)} \) conforms to its specification

In this case, we have (Def.2): \( \forall(x,o) \in E \times I_{C_1} : H^{(1)}(x,o)=1 \)

Using \((E \times I_{C_2} \subseteq E \times I_{C_1})\), we have: \( \forall(x,o) \in E \times I_{C_2} : H^{(1)}(x,o)=1 \)

The relationships \((R)\) in theorem implies: \([H^{(2)}(x,o) \iff (1 \land H_{(SI, SO)}(x,o))] \)

Therefore, we must to test \( m^{(2)} \) relatively to \( H_{(SI, SO)} \) (Figure 5):

- The method \( m^{(2)} \) is not in conformity with the constraint \( H_{(SI, SO)} \)

We have in this case, the following result (Figure 5):  

**Theorem 3**:

If the overriding method \( m^{(2)} \) of the subclass \( C_2 \) is not in conformity relatively to its specific constraint \((P^{(2)}, Q^{(2)}, Inv^{(2)})\), then the method \( m^{(2)} \) is not in conformity with its global specification \((P^{(2)}, Q^{(2)}, Inv^{(2)})\).

**Proof**:

We assume that the method \( m^{(2)} \) does not satisfy its specific constraint \( H_{(SI, SO)} \):

\[ \exists(x_0, o_0) \in E \times I_{C_2} : H_{(SI, SO)}(x_0, o_0)=0 \]

Applying the relationship \((R)\): \([H^{(2)}(x_0, o_0) \iff (H^{(1)}(x_0, o_0) \land 0)] \iff [H^{(2)}(x_0, o_0) \iff 0] \]

and consequently: \( \exists(x_0, o_0) \in E \times I_{C_2} : H^{(2)}(x_0, o_0)=0 \)

This induces that \( m^{(2)} \) is not in conformity with its global specification (Def.2).

- The method \( m^{(2)} \) conforms to its specific constraint \( H_{(SI, SO)} \)

We have in this case, the following result (Figure 5):

**Theorem 4**:

If the overridden method \( m^{(1)} \) of the class \( C_1 \) conforms to its specification, and its similar method \( m^{(2)} \) in the subclass \( C_2 \) conforms to the specific constraint \( H_{(SI, SO)} \), we deduce that the overriding method \( m^{(2)} \) conforms to its global specification.

**Proof**:

In this case, we have: \( \forall(x,o) \in E \times I_{C_2} : H^{(1)}(x,o)=1 \) and \( H_{(SI, SO)}(x,o)=1 \)

The relationships \((R)\) in theorem implies: \([H^{(2)}(x,o) \iff (1 \land 1)] \)

This means: \( \forall(x,o) \in E \times I_{C_2} : H^{(2)}(x,o)=1 \)

We deduce that the overriding method \( m^{(2)} \) conforms to its global specification (Def.2).
5. Evaluation

We evaluate the correctness of our approach by implementing the algorithm of conformity testing for inheritance. We consider for example of the conformity testing the methods `withdraw(1)` and `withdraw(2)` of the class `Account1` and `Account2` (Figure 6).

```java
class Account1
{
    protected double bal; /* bal is the account balance */
    public Account1(double x1)
    {
        this.bal=x1;
    }
    public void withdraw (int x1)
    {
        this.bal=this.bal - x1;
    }
}

class Account2 extends Account1
{
    private double InterestRate;
    public Account2(double x1, double x2)
    {
        super(x1);
        this.InterestRate=x2;
    }
    public void withdraw (int x1)
    {
        super.withdraw(x1);
        if (x1>bal)
        {
            this.bal=this.bal-(this.InterestRate)*x1;
            InterestRate = InterestRate/2;
        }
    }
}
```

**Figure 6. Account1 and Account2 Classes**
The constraints $H^{(1)}$ and $H^{(2)}$ of $\text{withdraw}^{(1)}$ and $\text{withdraw}^{(2)}$ in an algebraic specification are shown in the figure 7 ($x=x_1, o_a$ and $o_b$ are respectively the object $o$ after and before the call of the method):

We test firstly the similarity of $\text{withdraw}^{(2)}$ and $\text{Withdraw}^{(1)}$: for each $(x, o)$ that satisfy the common precondition of $\text{withdraw}^{(1)}$ and $\text{Withdraw}^{(2)}$ ($P^{(1)}(x,o)=1$), the condition of the block $\text{if...}$(method $\text{Withdraw}^{(2)}$ in Figure 6) is not satisfied, and thus the block $\text{if...}$ is not executed, this means that the method $\text{withdraw}^{(2)}$ does exactly the same thing as the method $\text{Withdraw}^{(1)}$(Figure6).As a result thereof, $\text{withdraw}^{(2)}$ and $\text{Withdraw}^{(1)}$ are similar relatively to the common specification ($P^{(1)}, Q^{(1)}, \text{Inv}^{(1)}$).

The second test concerns the conformity testing of $\text{withdraw}^{(2)}$ that is based on the conformity testing of $\text{withdraw}^{(1)}$(Figure 5):

- Conformity Testing for $\text{withdraw}^{(1)}$:
  In order to test the conformity of $\text{Withdraw}^{(1)}$ in class $\text{Account1}$, we generate randomly $x_1$ and the balance values in the interval $[-200, 200]$ with $N=100$ (Table1):

Table 1. Result of a Conformity Test of the $\text{Withdraw}^{(1)}$ Method

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>$x = x_1$</th>
<th>$O$</th>
<th>$P^{(1)}(x,o)$</th>
<th>$H^{(1)}(x,o)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>37</td>
<td>Account1(83)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>Account1(104)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>79</td>
<td>Account1(167)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>98</td>
<td>49</td>
<td>Account1(116)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>25</td>
<td>Account1(74)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>81</td>
<td>Account1(188)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The test result shows that for 100 iterations the constraint $H^{(1)}$ is always true ($H^{(1)} = 1$), this leads to the conclusion that the Withdraw$^{(1)}$ method is valid (Table 1). In this case it is necessary to test the method withdraw$^{(2)}$ relatively to its own constraint $H_{(S,SO)}$ (Figure 5).

- Conformity Testing for withdraw$^{(2)}$ relatively to $H_{(S,SO)}$:

In order to test the method withdraw$^{(2)}$ relatively to the constraint $H_{(S,SO)}$, we use an analysis with proof. The testing by proof of the method withdraw$^{(2)}$ relatively to the constraint $H_{(S,SO)}$ is used to strengthen the randomly testing. Indeed, we must have for satisfying the specific output (SO):

- The specific post-condition must be satisfied ($Q_2$').
- The specific invariant must be satisfied ($Inv_2$').

The constraint $Q_2$' is always satisfied, however we must proof that $Inv_2$' is satisfied.

For each created object $o_0$, we have: $0 < InterestRate_0 \leq 0.3$ where $InterestRate_0$ is the initial value assigned to $InterestRate$ when creating the object $o_0$ and $InterestRate_{(n)}$ is the value of $InterestRate$ after $n$ operations of type Withdraw$^{(2)}$ in an execution sequence.

We have: $InterestRate_{(n)} = InterestRate_{(n-1)} / 2, n \geq 1$ where $n$ is number of withdrawals (Figure 6).

The geometric series proposed is written in the general case as follows:

$$InterestRate_{(n)} = \left\lfloor InterestRate_{(0)} / 2^n \right\rfloor$$

with $n \geq 0$ and $0 < InterestRate_{(0)} \leq 0.3$)

We deduce that: $[\forall n : (0 < InterestRate_{(n)} \leq 0.3)]$ And consequently, the specific invariant is always satisfied (Figure 7). This leads to the conclusion that the method Withdraw$^{(2)}$ is in conformity to $H_{(S,SO)}$ and we can deduce that Withdraw$^{(2)}$ is in conformity with its global specification (Figure 5).

6. Conclusion

This paper introduces an approach to reduce the test sequences used for testing the conformity of an overriding method during inheritance operation in object-oriented models. The key idea of this work is the use of a technique that generates test data by exploiting the existing test sequences. Indeed, for the sub-classes methods that have the same behavior as their corresponding methods in a parent class, it is possible to reuse the data extracted from the conformity testing of an overridden method for testing the conformity of its overriding method.

We present firstly the relationship between the test model of overridden methods and overriding methods, and we show how the use of existing test sequences can make the generation of the test data during inheritance less expensive. Secondly, we present how it is possible to exploit the test result provided by the original class to test the methods in a sub-class.

Our future works are oriented to develop and generalize the formal model of conformity as defined in this paper for introduce the concept of methods security in inheritance based on the partition of invalid domains.

References


