International Journal of Software Engineering and Its Applications
Vol.7, No.5 (2013), pp.209-220
http://dx.doi.org/10.14257/ijseia.2013.7.5.19

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 JMl [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$(x=(x_1, x_2, \dots, x_n))$and *o* is the receiver object) such that:

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

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 precondition *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).

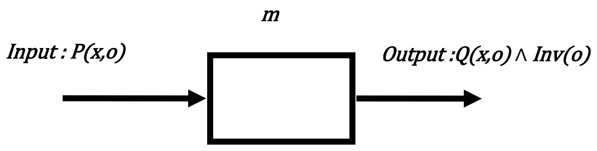

Figure 1. Specification of a Method m

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_1 such that *m* overrides a method of C_1. The original method and its overriding method in the subclass C_2 will be denoted respectively by $m^{(1)}$,$m^{(2)}$. ($P^{(1)}$,$Q^{(1)}$) denote respectively the pre-condition, the post-condition of the method $m^{(1)}$,and $Inv^{(1)}$ the invariant of C_1;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 $Q^{(1)}$of the overridden method ($Q^{(2)} \Rightarrow Q^{(1)}$),and the class invariant $Inv^{(2)}$ of the subclass C_2 must be equal to or stronger than the class invariant $Inv^{(1)}$ of the C_1 ($Inv^{(2)} \Rightarrow Inv^{(1)}$).

As a result of LSP:

The pre-condition $P^{(2)}$ of $m^{(2)}$ is the disjunction of $P^{(1)}$ and the specific pre-condition P_2' of $m^{(2)}$ (Figure 2): $P^{(2)} \Leftrightarrow (P^{(1)} \vee P_2')$ (1)

The post-condition $Q^{(2)}$ of $m^{(2)}$ is the conjunction of the post-condition $Q^{(1)}$ of $m^{(1)}$ and the specific post-condition $Q_2'$of $m^{(2)}$ (Figure 2): $Q^{(2)} \Leftrightarrow (Q^{(1)} \wedge Q_2')$ (2)

The invariant $Inv^{(2)}$ of the class C_2 is the conjunction of the invariant $Inv^{(1)}$ of the class C_1 and the specific invariant Inv_2' of C_2 (Figure 2): $Inv^{(2)} \Leftrightarrow (Inv^{(1)} \wedge Inv_2')$ (3)

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

$H^{(1)}(x,o) : P^{(1)}(x,o) \Rightarrow [Q^{(1)}(x, o) \wedge Inv^{(1)}(o)], (x,o) \in E \times I_{C1}$:The constraint of $m^{(1)}$.

$H^{(2)}(x,o) : P^{(2)}(x,o) \Rightarrow [Q^{(2)}(x,o) \wedge Inv^{(2)}(o)], (x,o) \in E \times I_{C2}$:The constraint of $m^{(2)}$.

Using *(1),(2),(3)* the constraint of $m^{(2)}$ will have the following form:

$H^{(2)} : [P^{(1)} \vee P_2'] \Rightarrow [Q^{(1)} \wedge Inv^{(1)} \wedge Q_2' \wedge Inv_2']$

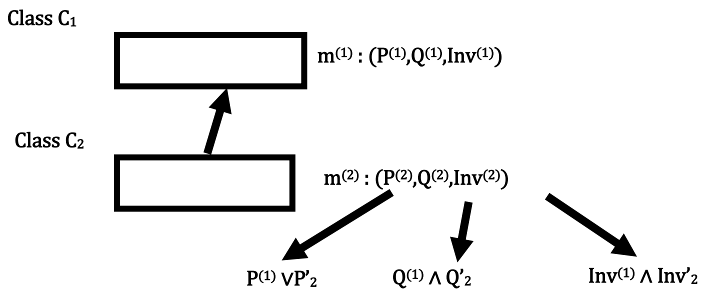

Figure 2. Constraints of $m^{(1)}$ and $m^{(2)}$

In our approach, the specification $(P^{(2)}, Q^{(2)}, Inv^{(2)})$ of $m^{(2)}$ 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 $m^{(2)}$.

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 $m^{(2)}$ 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 $m^{(2)}$.

Definition1: *(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 $(x,o) \in E \times I_{C2}$ such that:

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

Where I_{C2} 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: $H_{(BI\,,\,SO)}(x,o)$: $P^{(1)}(x,o) \Rightarrow [Q_2'(x,o) \wedge Inv_2'(o)]$

And $H_{(SI,\,BO)}(x,o)$: $P_2'(x,o) \Rightarrow [\ Q^{(1)}(x,o) \wedge Inv^{(1)}(o)\ ]$

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:

Theorem1:

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

Proof: Using the following result: $[(a \vee b) \Rightarrow (c \wedge d)] \Leftrightarrow [(a \Rightarrow c) \wedge (a \Rightarrow d) \wedge (b \Rightarrow c) \wedge (b \Rightarrow d)]$

We have: $[(P^{(1)} \vee P_2') \Rightarrow ((Q^{(1)} \wedge Inv^{(1)}) \wedge (Q_2' \wedge Inv_2'))] \Leftrightarrow [(P^{(1)} \Rightarrow (Q^{(1)} \wedge Inv^{(1)})) \wedge (P^{(1)} \Rightarrow (Q_2' \wedge Inv_2')) \wedge (P_2' \Rightarrow (Q^{(1)} \wedge Inv^{(1)})) \wedge (P_2' \Rightarrow (Q_2' \wedge Inv_2'))]$

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

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)* (Figure4).

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

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

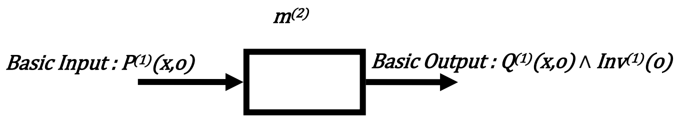
$m^{(2)}$
Basic Input : $P^{(1)}(x,o)$
Basic Output : $Q^{(1)}(x,o) \wedge Inv^{(1)}(o)$

Figure 3. Constraint BI-BO of an Overriding Method

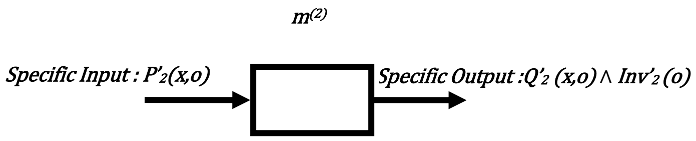
$m^{(2)}$
Specific Input : $P'_2(x,o)$
Specific Output : $Q'_2(x,o) \wedge Inv'_2(o)$

Figure 4. Constraint SI-SO of an Overriding Method

The constraints $H^{(1)}, H_{(SI\,,\,SO)}$,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 super-class.

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 $(x,o) \in E\times I_C$, the constraint *H* is satisfied : $\forall (x,o)\in E\times I_C : H(x,o)=1$

In other words, for a valid method: $\forall(x,o) \in E\times I_C$: If *(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:

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.

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 $H_{(SI\,,\,SO)}$ (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):

Theorem2:

If the overridden method m(1) in parent class C1 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: $\exists(x_0, o_0')\in E\times I_{C2} : H^{(1)}(x_0, o_0')=0$

We apply the relationship *(R)* (theorem1) on the pair (x_0, o_0') of the domain $E\times I_{C2}$:

$$(R)\Leftrightarrow[H^{(2)}(x_0,o_0') \Leftrightarrow(0\wedge H_{(SI,\,SO)}(x_0,o_0'))]\Leftrightarrow[H^{(2)}(x_0,o_0')\Leftrightarrow 0]$$

This means: $\exists(x_0, o_0') \in E\times I_{C2}$: $H^{(2)}(x_0, o_0')=0$

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) : $\forall(x, o)\in E\times I_{C1}$: $H^{(1)}(x, o)=1$

Using $(E\times I_{C2} \subset E\times I_{C1})$, we have: $\forall(x, o)\in E\times I_{C2}$: $H^{(1)}(x, o)=1$

The relationships (R) in theorem1implies:$[H^{(2)}(x,o) \Leftrightarrow (1 \wedge H_{(SI,\,SO)}(x, o))]\Leftrightarrow[H_{(SI,\,SO)}(x,o)]$

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):

Theorem3:

If the overriding method $m^{(2)}$ of the subclass C_2 *is* not in conformity relatively to its specific constraints(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 $H_{(SI,SO)}$:

$$\exists(x_0, o_0)\in E\times I_{C2}: H_{(SI,SO)}(x_0, o_0)=0$$

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

and consequently:$\exists(x_0,o_0) \in E\times I_{C2}$:$H^{(2)}(x_0,o_0)=0$

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):

Theorem4:

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 :$\forall(x, o)\in E\times I_{C2}$:$H^{(1)}(x, o)=1$ and $H_{(SI,SO)}(x, o)=1$

The relationships (*R*) in theorem1 implies:$[H^{(2)}(x, o) \Leftrightarrow(1\wedge 1)]$

This means : $\forall(x,o)\in E\times I_{C2}$: $H^{(2)}(x,o)=1$

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

```
If (m^(1) is in conformity relatively to (P^(1),Q^(1),Inv^(1)))
    If (m^(2) is in conformity relatively to (P_2',Q_2',Inv_2'))
        - m^(2) is in conformity relatively to (P^(2),Q^(2),Inv^(2))
    Else
        - m^(2) is not in conformity relatively to (P^(2),Q^(2),Inv^(2))
    EndIf
Else
    - m^(2) is not in conformity relatively to (P^(2),Q^(2),Inv^(2))
EndIf
```

Figure 5. Conformity Test Cases for an Overriding Method

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).

```
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 *withdraw*$^{(1)}$ and *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):

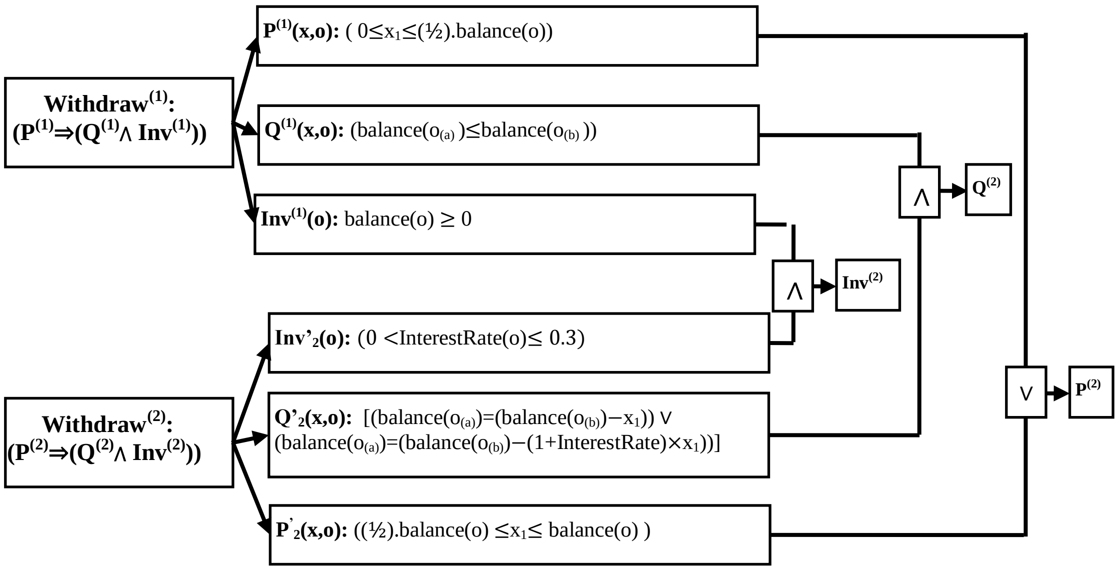
Withdraw$^{(1)}$: $(P^{(1)} \Rightarrow (Q^{(1)} \wedge Inv^{(1)}))$
$P^{(1)}(x,o)$: $(0 \leq x_1 \leq (½).balance(o))$
$Q^{(1)}(x,o)$: $(balance(o_{(a)}) \leq balance(o_{(b)}))$
$Inv^{(1)}(o)$: $balance(o) \geq 0$
Withdraw$^{(2)}$: $(P^{(2)} \Rightarrow (Q^{(2)} \wedge Inv^{(2)}))$
$Inv'_2(o)$: $(0 < InterestRate(o) \leq 0.3)$
$Q'_2(x,o)$: $[(balance(o_{(a)})=(balance(o_{(b)})-x_1)) \vee (balance(o_{(a)})=(balance(o_{(b)})-(1+InterestRate)\times x_1))]$
$P'_2(x,o)$: $((½).balance(o) \leq x_1 \leq balance(o))$
$\wedge$
$Q^{(2)}$
$\wedge$
$Inv^{(2)}$
$\vee$
$P^{(2)}$

Figure 7. Constraints of Withdraw$^{(1)}$ and Withdraw$^{(2)}$

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

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

- Conformity Testing for *withdraw* $^{(1)}$:

In order to test the conformity of *withdraw* $^{(1)}$ in class *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 Withdraw$^{(1)}$ Method

Iteration number:	$x = x_1$	*O*	$P^{(1)}(x,o)$	$H^{(1)}(x,o)$
1	37	Account1(83)	1	1
2	42	Account1(104)	1	1
3	79	Account1(167)	1	1
…	…	…	…	…
….	….	….	….	….
…..	…..	…..	…..	…..
98	49	Account1(116)	1	1
99	25	Account1(74)	1	1
100	81	Account1(188)	1	1

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 $H_{(SI,SO)}$ (Figure 5).

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

In order to test the method *withdraw*$^{(2)}$ relatively to the constraint $H_{(SI,SO)}$,we use an analysis with proof. The testing by proof of the method *withdraw*$^{(2)}$ relatively to the constraint $H_{(SI,SO)}$ 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:

$$InterestRate_{(n)} = [\ InterestRate_{(0)} /2^{\ n}\ ] \text{ with } n \geq 0 \text{ and} (\ 0 < InterestRate_0 \leq 0.3\ )$$

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 $H_{(SI,SO)}$,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

[1] B. K. Aichernig and P. A. P. Salas, “Test case generation by OCL mutation and constraint solving”, Proceedings of the International Conference on Quality Software, Melbourne, Australia, **(2005)** September 19-20, pp. 64-71.

[2] F. Bouquet, F. Dadeau, B. Legeard and M. Utting, “Symbolic animation of JML specifications”, International Conference on Formal Methods, LNCS, Springer-Verlag, vol. 3582, **(2005)** July, pp. 75-90.

[3] M. Benattou, J.-M. Bruel and N. Hameurlain, “Generating Test Data from OCL Specification”, Proceedings of the ECOOP'2002 Work-shop on Integration and Transformation of UML models (WITUML'2002), **(2002)**.

[4] K. Benlhachmi, M. Benattou and J.-L. Lanet, “Génération de Données de Test Sécurisé à partir d’une Spécification Formelle par Analyse des Partitions et Classification”, Proceedings of the 6th International Conference on Network Architectures and Information Systems Security (SAR-SSI 2011), La rochelle, France, **(2011)** May 18-21, pp. 143-150.

[5] K. Benlhachmi and M. Benattou, “A Formal Model of Similarity Testing for Inheritance in Object-Oriented Software”, Proceedings of the 2012 edition of the IEEE International Conference (CIST’2012), Fez, Morocco, **(2012)** October 24-26, pp. 38-42.

[6] K. Benlhachmi and M. Benattou, “Similarity Testing by Proof and Analysis of Partition for Object Oriented Specifications”, Journal of Theoretical and Applied Information Technology, vol. 46, Issue 1, no. 11, **(2012)** December, pp. 461-470.

[7] Y. Cheon and C. E. Rubio-Medrano, “Random Test Data Generation for Java Classes Annotated with JML Specifications”, Proceedings of the 2007 International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, vol. 2, **(2007)** June 25-28, pp. 385-392.

[8] G. T. Leavens, “JML’s Rich, Inherited Specification for Behavioral Subtypes”, Department of Computer Science Iowa State University, **(2006)** August 11.

[9] R. Bruce Findler, M. Latendresse and M. Felleisen, “Behavioral Contracts and Behavioral Subtyping”, Foundations of Software Engineering, Rice University, FSE, **(2001)**.

[10] B. H. Liskov and J. M. Wing, “A behavioral notion of subtyping”, MIT Laboratory for Computer Science, Carnegie Mellon University, ACM Transactions on Programming Languages and Systems, vol. 16, no. 6, **(1994)** November, pp. 1811-1841.

[11] G. T. Leavens and K. Kishore Dhara, “Concepts of Behavioral Subtyping and a Sketch of their Extension to Component-Based Systems”, G. T. Leavens and M. Sitaraman, editors, Foundations of Component-Based Systems, **(2000)**.

[12] B. H. Liskov and J. Wing, “Behavioral subtyping using invariants and constraints”, Technical Report CMU CS-99-156, School of Computer Science, Carnegie Mellon University, **(1999)** July.

[13] B. Meyer, “Object-oriented Software Construction”, Prentice Hall, **(1988)**.