IADIS International Journal on Computer Science and Information Systems
Vol. 9, No. 1, pp. 30-42
ISSN: 1646-3692

SIMILAR BEHAVIORS AND CONFORMITY TESTING IN INHERITANCE FOR AN OBJECT ORIENTED MODEL

Khalid Benlhachmi.*Laboratory of Research in Computer Science and Telecommunications.Faculty of Science, Ibn Tofail University Kenitra, 14000. Morocco.*

Mohammed Benattou.*Laboratory of Research in Computer Science and Telecommunications.Faculty of Science, Ibn Tofail University Kenitra, 14000. Morocco.*

ABSTRACT

The approach of this paper proposes a new concept of test which represents a way to compare the behaviors of methods in sub-classes and their original versions in the super-classes for an object oriented specification. The test process gives the conditions where the comparison can induce a similar behavior. The result of this test constitutes a solid basis to reuse the inherited specifications in the sub-classes for testing the conformity of overriding methods. 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 conformity testing in sub-classes is based on the similarity testing between overriding and overridden methods. 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, test data generation

1. INTRODUCTION

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 rel+ations 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 [B.k and P.A.P ,2005]and JMl [F.B and F.D and B.L and M.U , 2005] , and are used for generating test data from formal specifications [Mohammed B and Jean-Michel and Nabil;2002].

In this context, this paper introduces an approach for testing the conformity of an overriding method in object oriented models. The main idea of this work is the use of the similarity concept for testing the compatibility between overriding and overridden methods. 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 similarity and conformity of an overriding method by using the logical relationship between methods in subclasses and the original methods in the super classes.

The work presented in this paper allows to extend the constraint model defined in [Khalid B and M.B and Jean-louis L ,2011] for modeling the specification of an overriding method in subclass using inheritance

principle. This work is based on our model of similarity concept [Khalid Benlhachmi and M.Benattou,2012] for testing the conformity of overriding methods from conformity results of the original methods.

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 similarity and conformity constraints, in section 4 we define the matrix partitions deduced from the formal model and show how these partitions can be used to test the similarity of the overriding methods, in section 5 we show how we can generate test data for testing the similarity of methods, and in section 6 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 a parent class. Finally, we describe our approach with an example of similarity and 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 [Khalid B and M.B and Jean-louis L ,2011], 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 [Khalid Benlhachmi and M.Benattou,2012], we have developed a basic model for the concept of methods similarity, the test is based on a random generation of input data. In [Yoonsik C and Carlos E ,2007], 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 [Gray T ,2006], 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 [Robet B and Mario L and M.F,2001] 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 [Barbara H ans J.M ,1994], 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 [Gray T and Kristina K ,2000], 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. The work presented in this paper proposes a new approach for testing the overriding methods in subclasses using their basic specifications in super-classes, and allows specifying system behavior by constructing a model in terms of mathematical constructs.

3. FORMAL MODEL OF CONSTRAINT

This section presents a formal model of the generalized constraint defined in [Khalid B and M.B and Jean-louis L ,2011] which provides a way for modeling the specification of an overriding method by inheritance from a super-class. Indeed, firstly, we establish a series of theoretical concepts in order to create a solid basis for comparing the behavior of overriding methods in a subclass and the behavior of the original methods in the super-class. Secondly, we find the relationship between the test model of methods in subclasses and the test model in super-classes when the comparison of methods induces a similar behavior.

3.1 Formal model of constraint for a basic class

We have presented in [Khalid B and M.B and Jean-louis L ,2011] the definition of a formal model of constraint, illustrating the relationship between the pre-condition, the post-condition of a method and the invariant of the class. We use the same notation used in [Khalid B and M.B and Jean-louis L ,2011]: *P* is the precondition of the method *m* , *Q* is the post-condition of the same method and *Inv* is the invariant of the class *C*.

Let *C* be a class, and *m* be a method of *n* arguments $x=(x_1, x_2, ..., x_n)$. We define for each argument x_i its domain of values E_i .We denote $E=E_1 \times E_2 \times ... E_n$ the domain of input vector of the method *m*. We have defined in [Khalid B and M.B and Jean-louis L ,2011] the generalized constraint *H* of a method *m* of class *C* as a logical property of the pair *(x,o)* with *x* the vector of parameters and *o* 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* .

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), therefore, the objects used at the input of the method 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).

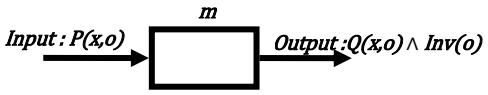

Figure1. Specification of a method m

3.2 Formal model and constraint propagation in inheritance

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 [Liskov B and J.Wing ,1988] and Meyer [Meyer B ,1988] 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:

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

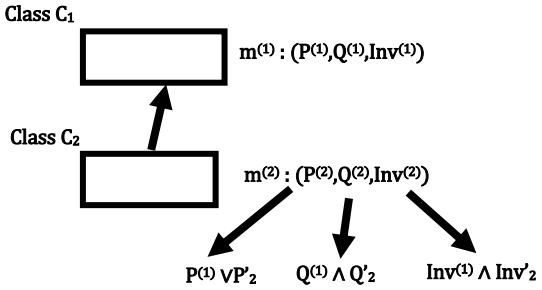

Figure 2. Constraints of $m^{(1)}$ and $m^{(2)}$

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)*:

- *Constraint $BI - SO$ of an overriding method:*

We propose the definition of a constraint $H_{(BI,\,SO)}$ allowing a partial view of the overriding method $m^{(2)}$ in the class C_2 . This constraint specifies the logical relationship between the input predicate $P^{(1)}$ inherited from $m^{(1)}$ and the output predicates (Q_2', Inv_2') specific to $m^{(2)}$.

Definition1: (Constraint $H_{(BI,\,SO)}$)

We define the constraint $H_{(BI,\,SO)}$ 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: $H_{(BI,\,SO)}(x,o) : P^{(1)}(x,o) \Rightarrow [Q_2'(x,o) \wedge Inv_2'(o)]$

With I_{C2} is the set of instances of C_2 ,and E is the vector of parameters of $m^{(2)}$.

- *Constraint $SI - BO$ of an overriding method:*

In the same way, we propose the definition of the constraint $H_{(SI,\,BO)}$.

Definition2: (Constraint $H_{(SI,\,BO)}$)

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

- *Constraint $SI - SO$ of an overriding method:*

In the same way, we propose the definition of the constraint $H_{(SI,\,SO)}$.

Definition3: (Constraint $H_{(SI,\,SO)}$)

We define the constraint $H_{(SI,\,SO)}$ of a 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: $H_{(SI,\,SO)}(x,o) : P_2'(x,o) \Rightarrow [Q_2'(x,o) \wedge Inv_2'(o)]$

3.3 Formal model of constraint for similar methods

We propose in this section the definition of a new concept to study the compatibility between the overriding methods and overridden methods in the super-class.

The objective is to find a relationship between the constraint $H^{(2)}$ of $m^{(2)}$ and the constraint $H^{(1)}$ of $m^{(1)}$ using the particular constraint $H_{(BI,SO)}$,$H_{(SI,BO)}$,$H_{(SI,SO)}$.

The two constraints : $H^{(1)}$: $(P^{(1)} \Rightarrow (Q^{(1)} \wedge Inv^{(1)}))$ and $H^{(2)} : (P^{(1)} \vee P_2') \Rightarrow (Q^{(1)} \wedge Inv^{(1)} \wedge Q_2' \wedge Inv_2')$ have common predicates: $(P^{(1)}, Q^{(1)}, Inv^{(1)})$ (Figure 3).

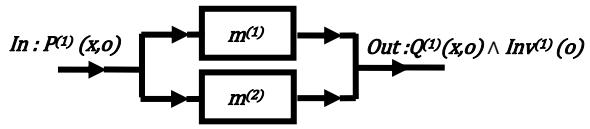

Figure 3. Common specification of $m^{(1)}$ and $m^{(2)}$

To determine the relationship between $H^{(1)}$ and $H^{(2)}$, we must analyze at first , the behavior of the two methods $m^{(1)}$ and $m^{(2)}$ relatively to the common specification (Figure 3). The methods $m^{(1)}$ and $m^{(2)}$ must have a similar behavior relatively to the specification $(P^{(1)}, Q^{(1)}, Inv^{(1)})$. Indeed, the original method in the basic class must evolve in subclasses with the condition that its behavior relatively to the basic specification is similar to the original version. Consequently, in Figure 4 we must have for each pair $(x,o)$of input :$a'=a$,$b'=b$ where a',a ,b',b are boolean values.

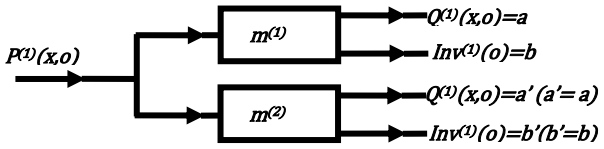

Figure 4. Condition of similarity

Definition 4 :(Similarity of methods)

A method $m^{(2)}$ of a class C_2 inheriting from the class C_1 is similar to the method $m^{(1)}$ relatively to the basic specification $(P^{(1)}, Q^{(1)}, Inv^{(1)})$ if :

-$m^{(2)}$ is an overriding method of $m^{(1)}$

-For each pair $(x,o) \in E \times I_{C2}$

$Q^{(1)}(x,o)$(respectively $Inv^{(1)}(o)$) has the same truth-value in output of $m^{(1)}$ and in output of $m^{(2)}$.

We consider two similar methods $m^{(1)}$ and $m^{(2)}$ in classes C_1 and C_2 : We have the logical equation between $H^{(2)}$ and $H^{(1)}$.

Theorem1:

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

Proof:

Using the following result such that a,b,c,d are logical predicates:

$$[(a \vee b) \Rightarrow (c \wedge d)] \Leftrightarrow [(a \Rightarrow c) \wedge (a \Rightarrow d) \wedge (b \Rightarrow c) \wedge (b \Rightarrow d)]$$

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:

$$[H^{(C2)} \Leftrightarrow (H^{(C1)} \wedge H_{(BI\,,\,SO)} \wedge H_{(SI\,,\,BO)} \wedge H_{(SI\,,\,SO)})]$$

The constraints $H^{(1)}$, $H_{(BI\,,\,SO)}$,$H_{(SI\,,\,BO)}$,$H_{(SI\,,\,SO)}$,and (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. SIMILARITY PARTITION

The analysis of the input domain of a method is a crucial step in the implementation of tests. Indeed, this analysis allows dividing the domains to locate the potential data which can affect the test problem and allows

to make a specific reasoning for each partition in order to identify the anomalies origin. In [Khalid B and M.B and Jean-louis L ,2011], we show how the constraint *H* may be used in the generation of domain partitions for each type of methods according to the classification proposed in [Yoonsik C and Carlos E ,2007]. This analysis of partition mainly concerns constructors and methods defined in a class without taking into account the inheritance relationships between classes.

The methods in subclass, during the operation of inheritance, are tested by a matrix partition: partition of similarity. This is a way to divide the input common domain of the two methods and to generate input data aimed at comparing the compatibility of an overriding method with the original version in a basic class relatively to their common specification.

For each input data *(x,o)* of the method $m^{(2)}$, we associate a square matrix of size 2 × 2 with boolean values (0 or1) which represents the four possible values for the quadruplet : (a ,b, a', b') (Figure 4).

Definition 5: (Similarity Application)

We define the Similarity application that associates each pair *(x,o)* of $E \times I_{C2}$ to its similarity matrix :

Similarity: $E \times I_{C2} \longrightarrow \mathcal{M}_2(\{0,1\})$

$(x,o) \rightarrow Similarity(x,o) = \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}$ where $s_{ij} \in \{0,1\}$.

s_{11} = *The truth-value of* $Q^{(1)}(x,o)$ *after the call of* $m^{(1)}$

s_{12} = *The truth-value of* $Inv^{(1)}(o)$ *after the call of* $m^{(1)}$

s_{21} = *The truth-value of* $Q^{(1)}(x,o)$ *after the call of* $m^{(2)}$

s_{22} = *The truth-value of* $Inv^{(1)}(o)$ *after the call of* $m^{(2)}$

The first row (s_{11}, s_{12}) of the matrix corresponds to the original method, and the second row (s_{21}, s_{22}) corresponds to the overriding method .

We deduce that $m^{(1)}$ and $m^{(2)}$ are similar only if: for each input data *(x,o)* the two rows of the matrix *Similarity (x,o)* are identical : $(s_{11}, s_{12}) = (s_{21}, s_{22})$

We divide $E \times I_{C2}$ on two subset *Sim* , *NotSim* :

The elements of *Sim* satisfy the constraint of similarity, but the elements of *NotSim* do not satisfy this constraint.

We divide *Sim* on four domains Sim_1, Sim_2, Sim_3 , Sim_4 (Figure 5):

$Sim_1 = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\}$

$Sim_2 = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}\}$

$Sim_3 = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}\}$

$Sim_4 = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}\}$

Then, we divide *NotSim* on $NotSim_1$ and $NotSim_2$:

The elements *(x,o)* of $NotSim_1$ do not satisfy the constraint of similarity and satisfy the condition :

$$[(Q^{(1)}(x,o) = 1 \text{ and } Inv^{(1)}(o) = 1) \text{ after the call of } m^{(2)}] .$$

This induces three possible cases corresponding to the following three parts of $NotSim_1$ (Figure 5):

$NotSim_{11} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}\}$

$NotSim_{12} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\}$

$NotSim_{13} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\}$

We can deduce that the elements of $NotSim_1$ satisfy the necessary condition:

$$\forall (x,o) \in E \times I_{C2} : [(x,o) \in NotSim_1] \Rightarrow [(Q^{(1)}(x,o) = 0 \text{ or } Inv^{(1)}(o) = 0) \text{ after the call of } m^{(1)}]$$

The elements *(x,o)* of $NotSim_2$ do not satisfy the constraint of similarity and constitute the other cases. This induces nine cases distributed on subsets of $NotSim_2$ (Figure 5):

$NotSim_{21} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}\}$

$NotSim_{22} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\}$

$NotSim_{23} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\}$

$NotSim_{24} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\}$

$NotSim_{25} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\}$

$NotSim_{26} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\}$

$NotSim_{27} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\}$

$NotSim_{28} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\}$

$NotSim_{29} = \{(x,o) \in E \times I_{C2} / Similarity(x,o) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\}$

We deduce that the elements of $NotSim_2$ satisfy the necessary condition:

$\forall (x,o) \in E \times I_{C2} : [(x,o) \in NotSim_2] \Rightarrow [(Q^{(1)}(x,o) = 0 \text{ or } Inv^{(1)}(o) = 0) \text{ after the call of } m^{(2)}]$

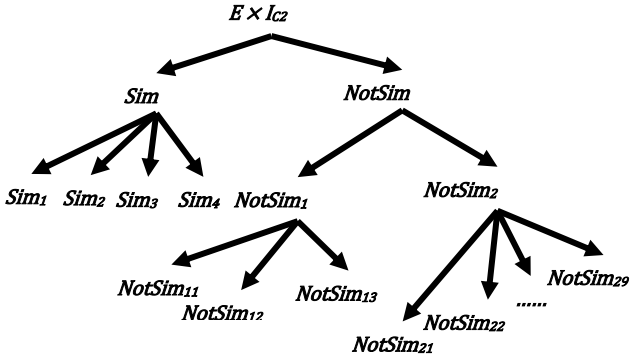

Figure 5. Similarity tree

5. SIMILARITY TESTING

The formal model of test proposed in [Khalid B and M.B and Jean-louis L ,2011] defines the notion of method validity in a basic class. This model is a way to generate test data for conformity. The testing process proposed in this section compares the compatibility between an overriding method in a subclass and its original version in the super-class. Our approach will be evaluated by implementing the algorithm of similarity testing for inheritance.

5.1 Algorithm of Similarity Testing

The similarity test generates random input data which satisfy the basic precondition of both methods ($P^{(1)}(x,o)=1$) and compares at the output of each method the behavior relatively to the common specification (Figure 6).

In this algorithm, we choose a threshold of test N and we generate randomly the pairs $(x,o) \in E \times I_{C2}$ which satisfy the basic precondition $P^{(1)}$, and for each (x,o) we compare the truth-value of the basic post-condition $Q^{(1)}$ for this (x,o) at the output of $m^{(1)}$ and $m^{(2)}$,at the same time we compare the truth-value of the basic invariant $Inv^{(1)}$ at the output of $m^{(1)}$ and $m^{(2)}$ .We assume that the objects used are generated from a valid constructor of the subclass, and the three sets Sim , $NotSim_1$, $NotSim_2$ are initialized to $\emptyset$ for each call of the algorithm. The test stops when we find a pair (x,o) for which the two rows of the matrix $Similarity(x,o)$ are not identical. In this case, the methods $m^{(1)}$ and $m^{(2)}$ are not similar relatively to the inherited specification. If we reach the threshold N of test without identifying a difference between the two rows for every matrix $Similarity$, we may admit with a rejected error margin (the limit N must be sufficiently large ($N \to \infty$)) that the methods $m^{(1)}$ and $m^{(2)}$ are similar.

```
do{
   do{
      for ( x_i in m^(2) parameter)
         {x_i = generate ( E_i);}
      x=(x_1,x_2,…,x_n);
      o = generate_object(C_2);
   }while(!P^(1)(x,o));
(x',o')=copy(x,o);
invoke"o. m^(1)(x)";
S_11=Q^(1)(x,o);S_12= Inv^(1)(o);
(x,o)=copy(x',o');
invoke"o. m^(2)(x)";
S_21=Q^(1)(x,o);S_22=Inv^(1)(o);
(x,o)=copy(x',o');
if((s_11,s_12)=(s_21,s_22 ) )
   Sim.add(x,o);
elseif((s_21,s_22)∈{(0,0),(0,1),(1,0 )})
   NotSim_2.add(x,o);
else
   NotSim_1.add(x,o);
     }while(Sim.size()<N && NotSim_1.isEmpty() && NotSim_2.isEmpty());
```

Figure 6. Similarity Test Algorithm of an Overriding Method

5.2 Evaluation

We evaluate the correctness of our approach by implementing the algorithm of similarity and conformity testing for inheritance. We consider for example of the similarity test the methods *withdraw* of the class *Account1* and *Account2* (Figure 7).

```
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 7. 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 8 ($x=x_1$,$o_{(a)}$ and $o_{(b)}$ are respectively the object o after and before the call of the method):

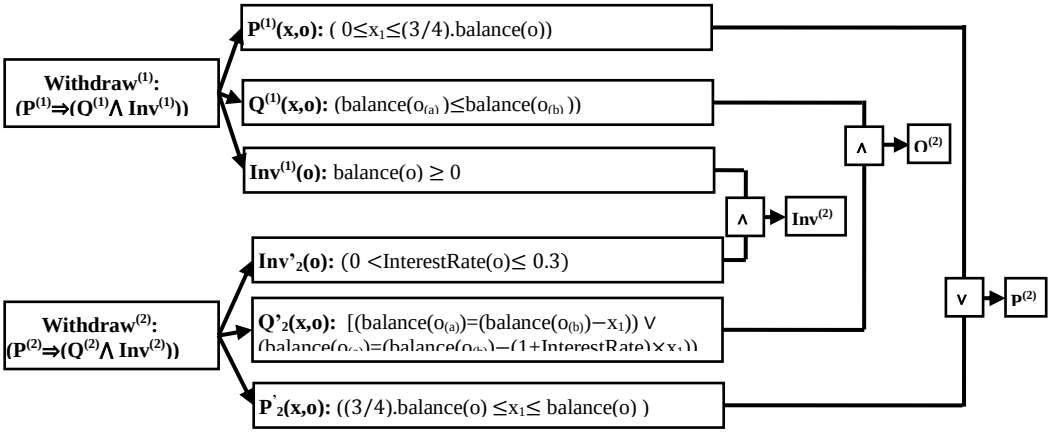

Figure 8. Constraints of withdraw$^{(1)}$ and withdraw$^{(2)}$

In order to test the similarity of *withdraw* methods in class *Account1* and *Account2* , we generate randomly x_1 and the balance values in the interval *(-200,200)* with the threshold limit *N =100* (Table 1)

Table 1. Result of a similarity test of the *Withdraw* methods

Iteration number:	$x = x_1$	*O*	$P^{(1)}(x,o)$	*(x,o)* ∈
1	*21*	*Account2(74,0.2)*	*1*	*Sim*
2	*45*	*Account2(130,0.1)*	*1*	*Sim*
3	*115*	*Account2(167,0.23)*	*1*	*Sim*
…	…	…	…	…
….	….	….	….	….
…..	…..	…..	…..	…..
98	*79*	*Account2(112,0.14)*	*1*	*Sim*
99	*52*	*Account2(87,0.11)*	*1*	*Sim*
100	*84*	*Account2(142,0.28)*	*1*	*Sim*

The test result shows that for 100 iterations the size of the set *Sim* is exactly the threshold limit of the test. We can conclude that the methods *withdraw*$^{(2)}$ and *Withdraw*$^{(1)}$ are similar relatively to the basic specification *($P^{(1)}$, $Q^{(1)}$, Inv_1)* (Table 1).

6. CONFORMITY TESTING

The formal model of test proposed in [Khalid B and M.B and Jean-louis L ,2011] defines the notion of method validity in a basic class. This model is 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. In [Khalid B and M.B and Jean-louis L ,2011], the conformity model is applied to constructors and methods of a basic class. The purpose of this section is to generalize the model of [Khalid B and M.B and Jean-louis L ,2011],[Khalid Benlhachmi and M.Benattou,2012] to test the conformity of an overriding method in a subclass.

6.1 Formal model of test for a basic class

In [Khalid B and M.B and Jean-louis L ,2011], 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 6: (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*.

6.2 Formal model of test for a basic class

The purpose of this paragraph is to test the conformity of an overriding method relatively to its specification in a derived class using the elements of conformity test of its original method in the super-class. This requires a preliminary test of similarity that compares the behavior of the two methods relatively to their common specification. In the following, we use the theoretical model of section 3 to test the conformity of a method $m^{(2)}$ of a subclass from the test result of similarity and conformity of the basic method $m^{(1)}$.

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 $H_{(BI\,,\,SO)}$, $H_{(SI\,,\,BO)}$, $H_{(SI\,,\,SO)}$ (*Th.1*).

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 :

Theorem2:

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.6*) 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)* (*Th.1*) on the pair (x_0,o_0') of the domain $E \times I_{C2}$:

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

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

This shows (*Def.6*) 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.6*) : $\forall (x,o) \in E \times I_{C1} : H^{(1)}(x,o)=1$

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)* is not sufficient to inform us about the truth-value of $H^{(2)}$.This requires a test of $m^{(2)}$ relatively to the partial constraints $H_{(BI,SO)}$,$H_{(SI\,,\,BO)}$ and $H_{(SI\,,\,SO)}$ (Figure 9):

The test principle of $m^{(2)}$ relatively to the constraint $H_{(BI,SO)}$ is to generate randomly data that satisfy the basic precondition ($Basic\ Input$), and for each generated element we must evaluate the truth-value of the specific post-condition and invariant of $m^{(2)}$ ($Specific\ output$). The test stops when an element violates the constraint $H_{(BI,SO)}$. If the threshold of test is reached with $H_{(BI,SO)}$ satisfied, we admit with a negligible margin of error ($N \to \infty$) that the method $m^{(2)}$ is conform relatively to $H_{(BI,SO)}$.

In the same way, we can test $m^{(2)}$ relatively to the constraints $H_{(SI\,,\,BO)}$ and $H_{(SI\,,\,SO)}$. Indeed, the cases of conformity for an overriding method are specified in the Figure 9.

```
If (m(1) conforms to H(1))
  If (m(2) conforms to H(BI,SO))
    If (m(2) conforms to H(SI,BO))
      If (m(2) conforms to H(SI,SO))
        - m(2) conforms to H(2)
      Else
        - m(2) is not conform to H(2)
      EndIf
    Else
      - m(2) is not conform to H(2)
    EndIf
  Else
    - m(2) is not conform to H(2)
  EndIf
Else
  - m(2) is not conform to H(2)
EndIf
```

Figure 9. Conformity test cases for an overriding method

6.3 Formal model of test for a basic class

We consider for example of the conformity testing the overriding method *withdraw*$^{(2)}$ of the class *Account2* (Figure 7). The test of the last section shows that *withdraw* $^{(1)}$ and *withdraw*$^{(2)}$ are similar. For testing the conformity of *withdraw*$^{(2)}$, we test firstly the conformity of *withdraw*$^{(1)}$:

- 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* (Table 2):

Table 2- 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	*12*	*Account1(23)*	*1*	*1*
2	*26*	*Account1(41)*	*1*	*1*
3	*66*	*Account1(100)*	*1*	*1*
…	…	…	…	…
….	….	….	….	….
…..	…..	…..	…..	…..
98	*57*	*Account1(87)*	*1*	*1*
99	*19*	*Account1(54)*	*1*	*1*
100	*81*	*Account1(119)*	*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 2). In this case it is necessary to test the method *withdraw*$^{(2)}$ relatively to the constraint $H_{(BI,SO)}$.

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

In order to test the method *withdraw*$^{(2)}$ relatively to the constraint $H_{(BI,SO)}$,we use an analysis with proof .The testing by proof of the method *withdraw*$^{(2)}$ relatively to the constraint $H_{(BI,SO)}$ 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 7) .

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 8).This leads to the conclusion that the method $Withdraw^{(2)}$ is conform to $H_{(BI,SO)}$,and it is necessary to test $withdraw^{(2)}$ relatively to the constraint $H_{(SI,BO)}$ (Figure 9).

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

The result of the randomly test of the method $withdraw^{(2)}$ relatively to $H_{(SI,BO)}$:

Table 3. Result of a conformity test of the $Withdraw^{(2)}$ relatively to $H_{(SI,BO)}$

Iteration number:	$x = x_1$	*O*	$P_2'(x,o)$	$H_{(SI,BO)}(x,o)$
1	*38*	*Account2(49,0.15)*	*1*	*1*
2	*47*	*Account2(54,0.05)*	*1*	*1*
3	*103*	*Account2(134,0.19)*	*1*	*1*
4	*40*	*Account2(44,0.07)*	*1*	*1*
5	*137*	*Account2(177,0.18)*	*1*	*1*
6	*22*	*Account2(23,0.25)*	*1*	*0*

As is shown in table 3, $Withdraw^{(2)}$ is not conform to $H_{(SI,BO)}$, the pair $(x,o)=(22, Account2\ (23,0.25))$ in the iteration 6 makes $H_{(SI,BO)}$ false $(H_{(SI,BO)} = 0\ )$.Consequently, the $Withdraw^{(2)}$ method is not conform to its global specification $(H^{(2)}(x,o)=0)$.

7. CONCLUSION

The approach of this paper proposes a new concept of test which represents a way to compare the behaviors of methods in sub-classes and their original versions in the super-classes for an object oriented specification. The test process gives the conditions where the comparison can induce a similar behavior. The result of this test constitutes a solid basis to reuse the inherited specifications in the sub-classes for testing the conformity of overriding methods.

We analyze firstly, how an overriding method can use the specification of its overridden method in the super-class, and we present the relationship between the test model of methods in a subclass and the basic version in the super-class. Secondly, we show how the conformity testing of methods in sub-classes can be deduced from their original methods. The principal idea of the proposed work is based on the partition of domains for specifying all cases of methods similarity and for proving formally the correctness of the model.

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

REFERENCES

Barbara 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, November 1994, Pages 1811-1841.

B. K. Aichernig and P. A. P. Salas. "Test case generation by OCL mutation and constraint solving", *In Proceedings of the International Conference on Quality Software*, Melbourne, Australia, September 19-20, 2005, pages 64–71, 2005.

F. Bouquet, F. Dadeau, B. Legeard, and M. Utting. "Symbolic animation of JML specifications",*In International Conference on Formal Methods*, volume 3582 of LNCS, pages 75– 90. Springer-Verlag, July 2005.

Gary T.Leavens ,"*JML's Rich,Inherited Specification for Behavioral Subtypes*", Department of Computer Science Iowa State University , August 11,2006.

Gary T.Leavens , Krishna Kishore Dhara , "*Concepts of Behavioral Subtyping and a Sketch of their Extension to Component-Based Systems*", In G. T. Leavens and M. Sitaraman, editors, Foundations of Component-Based Systems ,2000.

Khalid Benlhachmi, and M.Benattou."A Formal Model of Similarity Testing for Inheritance in Object-Oriented Software". *In Proceedings of the 2012 edition of the IEEE International Conference (CIST'2012)* ,October 24-26,2012, Fez, Morocco. Pages 38-42.

Khalid Benlhachmi, and M.Benattou, and Jean-louis Lanet."Génération de Données de Test Sécurisé à partir d'une Spécification Formelle par Analyse des Partitions et Classification*". In Proceedings of the 6 th International Conference on Network Architectures and Information Systems Security (SAR-SSI 2011)* ,May 18-21,2011,La rochelle ,France. Pages 143-150.

Liskov, B. H. and J. Wing."*Behavioral subtyping using invariants and constraints*", Technical Report CMU CS-99-156, School of Computer Science, Carnegie Mellon University, July 1999.

Meyer, B. "*Object-oriented Software Construction*". Prentice Hall, 1988.

Mohammed Benattou, Jean-Michel Bruel, and Nabil Hameurlain. "Generating Test Data from OCL Specification",*In Proceedings of the ECOOP'2002 Work-shop on Integration and Transformation of UML models (WITUML'2002)*, 2002.

Robet Bruce Findler , and Mario Latendresse , and Matthias Felleisen ,"Behavioral Contracts and Behavioral Subtyping", Foundations of Software Engineering , Rice University , FSE 2001.

Yoonsik Cheon and Carlos E. Rubio-Medrano."Random Test Data Generation for Java Classes Annotated with JML Specifications", *In Proceedings of the 2007 International Conference on Software Engineering Research and Practice*, Volume II, June 25-28, 2007, Las Vegas, Nevada, pages 385-392