Journal of Information Security, **2013, 4, 113-123**
http://dx.doi.org/10.4236/jis.2013.42013 Published Online April 2013 (http://www.scirp.org/journal/jis)

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

Khalid Benlhachmi, Mohammed Benattou
Laboratory of Research in Computer Science and Telecommunications,
Faculty of Science, Ibn Tofail University, Kenitra, Morocco
Email: benlhachmi11@yahoo.fr

Received March 1, 2013; revised April 1, 2013; accepted April 9, 2013

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 first 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. In this context the 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 second objective is the use of invalid data which do not satisfy the precondition constraint and induce valid output values for introducing a new concept of test called secure testing. The implementation of this approach is based on a random generation of test data and analysis by formal proof.

Keywords: Conformity Test; Security Test; Constraints Resolution; Formal Specification; Inheritance

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]. The proposed test oracles in the industry are satisfied with the conformity test methods, and are carried out by generating test data that conform to the pre-condition constraint. However, a false pre-condition may induce both valid post-condition and invariant at the same time. A correctly implemented testing method should eliminate cases of invalid data which lead to valid results.

In this context, this paper introduces in the first instance 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 second contribution of our approach is the definition of a security testing based on the generation of invalid input data which induce valid output values. Indeed, in this paper we show how we can use the valid data extracted from the pre-condition constraint to test the conformity of an overriding method, and how the data which does not satisfy the precondition can be used to test the security of similar methods. Our test oracle enables to detect anomalies in invalid inputs that imply valid results.

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 and the security of overriding methods from test 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, and to integrate the invalid data in test process. 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 invalid data that do not satisfy the precondition constraint and induce valid post-conditions.

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 data for testing the conformity of overriding methods during inheritance operation, in Section 5 we present our approach of security testing that strengthens the conformity testing, and we show how the security testing of an overriding method can be deduced from its overridden method in parent class. Finally, in Section 6 we describe our approach with an example of conformity and security 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. The main difference between our work and other related works is the definition of the security testing. Indeed, the test methods have focused only on valid inputs satisfying the pre-condition, and do not integrate the invalid data in test process. Our approach shows how we can use the valid and invalid data extracting from the pre-condition to test the conformity and the security of overridden and overriding methods.

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 in-

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

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

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**).

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

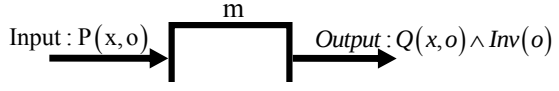

Figure 1. Specification of a method *m*.

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 $\left(P^{(1)} \Rightarrow P^{(2)}\right)$, the post-condition $Q^{(2)}$ is stronger than the post-condition $Q^{(1)}$ of the overridden method $\left(Q^{(2)} \Rightarrow Q^{(1)}\right)$, 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 $\left(Inv^{(2)} \Rightarrow Inv^{(1)}\right)$.

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**):

$$P^{(2)} \Leftrightarrow \left(P^{(1)} \vee P_2'\right) \quad (1)$$

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**):

$$Q^{(2)} \Leftrightarrow \left(Q^{(1)} \wedge Q_2'\right) \quad (2)$$

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

$$Inv^{(2)} \Leftrightarrow \left(Inv^{(1)} \wedge Inv_2'\right) \quad (3)$$

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

The constraint of $m^{(1)}$:

$$H^{(1)}(x,o): P^{(1)}(x,o) \Rightarrow \left[Q^{(1)}(x,o) \wedge Inv^{(1)}(o)\right],$$
$$(x,o) \in E \times I_{c_1}$$

The constraint of $m^{(2)}$:

$$H^{(2)}(x,o): P^{(2)}(x,o) \Rightarrow \left[Q^{(2)}(x,o) \wedge Inv^{(2)}(o)\right],$$
$$(x,o) \in E \times I_{c_2}$$

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

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

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

Two inputs: the Basic Input $\left(BI = P^{(1)}\right)$ and the Specific Input $\left(SI = P_2'\right)$ of $m^{(2)}$.

Two outputs: Basic Output $\left(BO = \left(Q^{(1)}, Inv^{(1)}\right)\right)$ and the Specific Output $\left(SO = \left(Q_2', Inv_2'\right)\right)$ 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

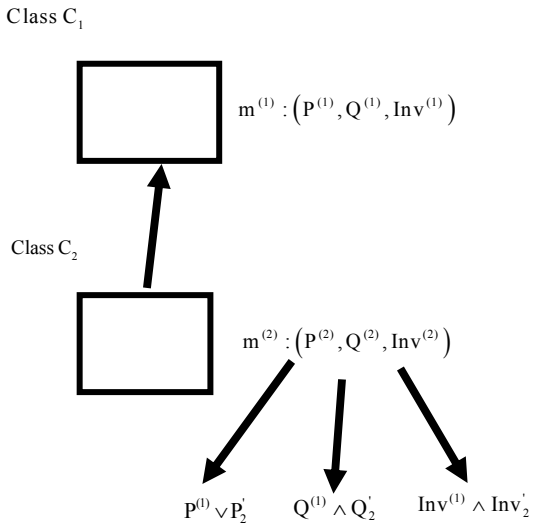

Figure 2. Constraints of $m^{(1)}$ and $m^{(2)}$.

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 $\left(P^{(1)}, Q^{(1)}, Inv^{(1)}\right)$. 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 $\left(Q_2', Inv_2'\right)$ 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 $(x,o) \in E \times I_{C_2}$ such that:

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

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

The logical implication in the proposed formula means that: each call of method 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.

In the same way, we put:

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

And

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

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:

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

Proof: Using the following result:

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

We have:

$$\left[\left(P^{(1)} \vee P_2'\right) \Rightarrow \left(\left(Q^{(1)} \wedge Inv^{(1)}\right) \wedge \left(Q_2' \wedge Inv_2'\right)\right)\right]$$
$$\Leftrightarrow \left[\left(P^{(1)} \Rightarrow \left(Q^{(1)} \wedge Inv^{(1)}\right)\right) \wedge \left(P^{(1)} \Rightarrow \left(Q_2' \wedge Inv_2'\right)\right)\right.$$
$$\left. \wedge \left(P_2' \Rightarrow \left(Q^{(1)} \wedge Inv^{(1)}\right)\right) \wedge \left(P_2' \Rightarrow \left(Q_2' \wedge Inv_2'\right)\right)\right]$$

Therefore, we have the following result:

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

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, (*R*) is equivalent to

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

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 no-

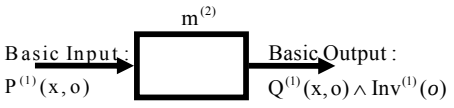

Figure 3. Constraint *BI-BO* of an overriding method.

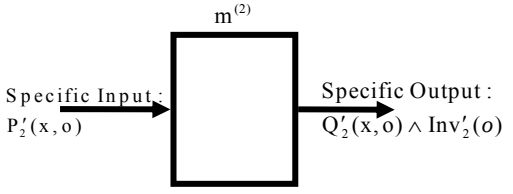

Figure 4. Constraint *SI-SO* of an overriding method.

tion 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 $\left(H(x,o)=0\right)$ 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 $\left(P^{(1)},Q^{(1)},Inv^{(1)}\right)$.

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 $H_{(SI,SO)}$ (*Theorem*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:

Theorem 2:

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

Proof:

We assume that the method $m^{(1)}$ is not in conformity with its specification. This means (*Def.* 2) that:

$$\exists(x_0,o_0)\in E\times I_c : H^{(1)}(x_0,o_0)=0$$

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*) (*Theorem* 1) on the pair (x_0,o_0') of the domain $E\times I_{C_2}$:

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

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 $\left(E\times I_{c_2}\subset E\times I_{c_1}\right)$, we have:

$$\forall(x,o)\in E\times I_{c_2} : H^{(1)}(x,o)=1$$

The relationships (*R*) in theorem1implies:

$$\left[H^{(2)}(x,o)\Leftrightarrow\left(1\wedge H_{(SI,SO)}(x,o)\right)\right]\Leftrightarrow\left[H_{(SI,SO)}(x,o)\right]$$

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

- $m^{(2)}$ *is not in conformity with* $H_{(SI,SO)}$

We have in this case, the following result:

Theorem 3:

If the overriding method $m^{(2)}$ of the subclass C_2 is not in conformity relatively to its specific constraints $\left(P_2',Q_2',Inv_2'\right)$, then the method $m^{(2)}$ is not in conformity with its global specification $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$.

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*):

$$\left[H^{(2)}(x_0,o_0)\Leftrightarrow\left(H^{(1)}(x_0,o_0)\wedge 0\right)\right]$$
$$\Leftrightarrow\left[H^{(2)}(x_0,o_0)\Leftrightarrow 0\right]$$

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) .

- $m^{(2)}$ *conforms to its specific constraint* $H_{(SI,SO)}$

We have in this case, the following result:

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:

$$\forall (x,o) \in E \times I_{c_2} : H^{(1)}(x,o) = 1 \quad \text{and} \quad H_{(SI,SO)}(x,o) = 1$$

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

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. Security Testing

5.1. Formal Model of Security Testing

The realized tests for conformity consider that the input data satisfy the precondition. However, a false pre-condition may induce both a post-condition and an invariant which are valid. A correctly testing of implementation of a method should reject cases of invalid data which provide valid results. Most of test oracles do not integrate the invalid data in test process. This section presents a complementary test aimed to study the invalid inputs of a method that conforms to its specification. Indeed, an anomaly difficult to be detected arises when a couple (*x*,*o*) does not satisfy the pre-condition *P* is accepted and induces both a post-condition *Q* and invariant *Inv* that are valid at the output. Our approach allows resolving these anomalies by the introduction of a complementary test for each method *m* that conforms to its specification. On the theoretical level, we are looking for strengthening the current constraint *H* in order to integrate this type of test.

Consider a method *m* of class *C* such that: *o* the receiver object and *x* the vector of parameters.

Definition 3: (*Secure method*)

The method *m* is secure relatively to its specification if it satisfies the following conditions:

- It conforms to its specification.
- For each invalid couple (*x*,*o*) of input (does not satisfy the pre-condition: $P(x,o) = 0$), the post-condition *Q* and the invariant *Inv* should not be both valid in output: $\left(Q(x,o) \wedge Inv(o) = 0\right)$.

As is shown in **Table 1**, the constraint *H* defined above for conformity test takes in the security test the following form:

$$\forall (x,o) \in E \times I_c : P(x,o) \Leftrightarrow \left[Q(x,o) \wedge Inv(o) \right]$$

We note:

$$H_{security}(x,o) : P(x,o) \Leftrightarrow \left[Q(x,o) \wedge Inv(o) \right]$$

Table 1. Truth cases of *H* and $H_{security}$.

P	*Q*	*Inv*	$H : P \Rightarrow (Q \wedge Inv)$	$H_{security} : P \Leftrightarrow (Q \wedge Inv)$
1	1	1	1	1
1	1	0	0	0
1	0	1	0	0
1	0	0	0	0
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0

In this test, we assume that with an invalid input, we can expect only an invalid result: any valid result coming from an invalid input indicates the presence of a security problem into the method implementation.

Theorem 5: The method *m* is secure relatively to its specification, if: $\forall (x,o) \in E \times I_c : H_{security}(x,o) = 1$.

5.2. Algorithm of Security Testing

The main aim of security testing algorithm is the use of the proposed security testing model for checking if the method under test is secure relatively to its specification. We assume that the method under test is conforming to its specification. The test data generation requires that the input values generated must not satisfy the pre-condition $(P = 0)$. The execution of each security test algorithm stops when the constraint $H_{security}$ becomes false $\left(H_{security}(x,o) = 0\right)$ or when we reach the threshold of test with $H_{security} = 1$. In the **Figure 5**, the constant *N* is an input value and (A_{21}, A_{22}, A_{23}, A_{24}) is the partition of the invalid input domain of the method *m*:

$$A_{21} = \left\{ (x,o) \in E \times I_c / \left(P(x,o), Q(x,o), Inv(o) \right) = (0,0,0) \right\}$$

$$A_{22} = \left\{ (x,o) \in E \times I_c / \left(P(x,o), Q(x,o), Inv(o) \right) = (0,0,1) \right\}$$

$$A_{23} = \left\{ (x,o) \in E \times I_c / \left(P(x,o), Q(x,o), Inv(o) \right) = (0,1,0) \right\}$$

$$A_{24} = \left\{ (x,o) \in E \times I_c / \left(P(x,o), Q(x,o), Inv(o) \right) = (0,1,1) \right\}$$

5.3. Security Testing in Inheritance

The purpose of this paragraph is to test the security of an overriding method $m^{(2)}$ relatively to its global specification $\left(P^{(2)}, Q^{(2)}, Inv^{(2)} \right)$ in a derived class C_2 using the


```
do{
      do{
for ( xi in set of parameters of m)
{xi=generate(Ei);}
x= (x1,x2,…,xn);o = generate_object();
        }while(P(x,o));
invoke"o.m(x)"
if( !Q(x,o)&& !Inv(o))
A21.add(x,o);
elseif( !Q(x,o)&&Inv(o))
A22.add(x,o);
elseif( Q(x,o)&& !Inv(o))
A23.add(x,o);
else A24.add(x,o)
}while(A21.size()<N && A22.size()<N && A23.size()<N
&& A24.isEmpty());
```

Figure 5. Security test algorithm of the method *m*.

elements of security test of its original method $m^{(1)}$ in the super-class C_1.

Definition 4: (*Security of* $m^{(2)}$ *relatively to* $\left(P^{(1)},Q^{(1)},Inv^{(1)}\right)$

An overriding method $m^{(2)}$ is secure relatively to its inherited specification $\left(P^{(1)},Q^{(1)},Inv^{(1)}\right)$ if:

- $m^{(2)}$ conforms to its inherited specification $\left(P^{(1)},Q^{(1)},Inv^{(1)}\right)$.
- For each couple (x,o) in the invalid input domain of $m^{(2)}$ $\left((P^{(1)}\vee P_2')(x,o)=0\right)$, the post-condition $Q^{(1)}$ and the invariant $Inv^{(1)}$ should not be both valid in output of $m^{(2)}$ $\left(Q^{(1)}(x,o)\wedge Inv^{(1)}(o)=0\right)$.

Definition 5: (*Security of an overriding method* $m^{(2)}$ *relatively to* $\left(P_2',Q_2',Inv_2'\right)$).

An overriding method $m^{(2)}$ is secure relatively to its own specification $\left(P_2',Q_2',Inv_2'\right)$ if:

- $m^{(2)}$ conforms to its own specification $\left(P_2',Q_2',Inv_2'\right)$.
- For each couple (x,o) in the invalid input domain of $m^{(2)}$ $\left(\left(P^{(1)}\vee P_2'\right)(x,o)=0\right)$, the post-condition Q_2' and the invariant Inv_2' should not be both valid in output of $m^{(2)}$ $\left(\left(Q_2'(x,o)\wedge Inv_2'\right)(o)=0\right)$.

Definition 6: (*Security of an overriding method* $m^{(2)}$ *relatively to its global specification* $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$)

An overriding method $m^{(2)}$ is secure relatively to its global specification $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$ if and only if:

- $m^{(2)}$ is secure relatively to its inherited specification $\left(P^{(1)},Q^{(1)},Inv^{(1)}\right)$.
- $m^{(2)}$ is secure relatively to its own specification $\left(P_2',Q_2',Inv_2'\right)$.

Theorem 6:

An overriding method $m^{(2)}$ is secure relatively to its global specification $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$ if:

- $m^{(2)}$ conforms relatively to its global specification $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$.
- For each invalid couple (x,o) of input (does not satisfy the pre-condition $\left(P^{(2)}(x,o)=0\right)$, the post-condition $Q^{(2)}$ and the invariant $Inv^{(2)}$ should not be both valid in output of $m^{(2)}$ $\left(Q^{(2)}(x,o)\wedge Inv^{(2)}(o)=0\right)$.

Theorem 7:

An overriding method $m^{(2)}$ (that conforms to its global specification) is not secure relatively to this specification if

$$\exists(x,o)\in E\times I_{C_2}:\left[\left(P^{(1)}\vee P_2'\right)(x,o)=0\right]$$

in input of $m^{(2)}$ and

$$\left[\left(\left(Q^{(1)}\wedge Inv^{(1)}\right)(x,o)=1\right)\vee\left(\left(Q_2'\wedge Inv_2'\right)(x,o)=1\right)\right]$$

in output of $m^{(2)}$.

The security test of $m^{(2)}$ relatively to its global specification $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$ is performed only if the conformity test of $m^{(2)}$ relatively to $\left(P^{(2)},Q^{(2)},Inv^{(2)}\right)$ is validated. We consider that $m^{(1)}$ and $m^{(2)}$ are in conformity with their specifications, and we assume that the security test of $m^{(2)}$ starts when the security testing of $m^{(1)}$ is completed. Indeed, the security test of $m^{(1)}$ induces two cases: $m^{(1)}$ is secure relatively to its specification or $m^{(1)}$ is not secure relatively to its specification:

- *Case* 1: $m^{(1)}$ is not secure relatively to its specification.

We assume that the method $m^{(1)}$ is not secure relatively to its specification. This means (*Def.* 3 *and Th.* 5) that:

$$\left(\exists(x_0,o_0)\in E\times I_{C_1}:H^{(1)}_{security}(x_0,o_0)=0\right)$$

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:

$$\left(\exists(x_0,o_0')\in E\times I_{C_2}:H^{(1)}_{security}(x_0,o_0')=0\right)$$

i.e.

$$\left[\left[P^{(1)}(x_0,o_0')=0 \text{ in input of } m^{(1)}\right] \text{ and } \left[\left(Q^{(1)}\wedge Inv^{(1)}\right)(x_0,o_0')=1 \text{ in output of } m^{(1)}\right]\right]$$

$m^{(1)}$ and $m^{(2)}$ are similar, then we have:

$$\left[\left[P^{(1)}(x_0,o_0')=0 \text{ in input of } m^{(2)}\right] \text{ and } \left[\left(Q^{(1)}\wedge Inv^{(1)}\right)(x_0,o_0')=1 \text{ in output of } m^{(2)}\right]\right]$$

We have two cases: $P_2'(x_0,o_0')=0$ or $P_2'(x_0,o_0')=1$

- $P_2'(x_0,o_0')=0$.

In this case, we have:

$$\left[\left[\left(P^{(1)}\vee P_2'\right)(x_0,o_0')=0 \text{ in input of } m^{(2)}\right] \text{ and } \left[\left(Q^{(1)}\wedge Inv^{(1)}\right)(x_0,o_0')=1 \text{ in output of } m^{(2)}\right]\right]$$

Consequently the method $m^{(2)}$ is not secure relatively

to $\left(P^{(1)}, Q^{(1)}, Inv^{(1)}\right)$ (*Def.* 4).

Finally, this shows (*Th.* 7) that the method $m^{(2)}$ is not secure relatively to its global specification $\left(P^{(2)}, Q^{(2)}, Inv^{(2)}\right)$.

- $P_2'\left(x_0, o_0'\right) = 1$

In this case, we have: $\left(\left(P^{(1)} \vee P_2'\right)\left(x_0, o_0'\right) = 1\right)$

Consequently the couple $\left(x_0, o_0'\right)$ is in the valid input domain: the security of $m^{(2)}$ cannot be deduced from the security test of $m^{(1)}$ (**Figure 6**).

- *Case* 2: $m^{(1)}$ is secure relatively to its specification.

In this case, we must test the method $m^{(2)}$ relatively to its own specification $\left(P_2', Q_2', Inv_2'\right)$ (**Figure 6**).

6. Evaluation

We evaluate the correctness of our approach by implementing the algorithm of conformity and security testing for inheritance. We consider for example of conformity and security testing the methods *withdraw*$^{(1)}$ and *withdraw*$^{(2)}$ of the class *Account*1 and *Account*2 (**Figure 7**).

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):

- Conformity testing for *withdraw*$^{(2)}$ and *withdraw*$^{(1)}$

We test firstly the similarity of *withdraw*$^{(2)}$ and *withdraw*$^{(1)}$ on the common valid domain *CVD* $\left(CVD = \left\{(x,o) \in E \times I_{C_2} \,/\, P^{(1)}(x,o) = 1\right\}\right)$: for each (x,o) that satisfy the common precondition of *withdraw*$^{(1)}$ and *withdraw*$^{(2)}$ $\left(P^{(1)}(x,o) = 1\right)$, the condition of the block if {…} (method *withdraw*$^{(2)}$ in **Figure 7**) is not satisfied, and thus the block if {…} is not executed, this means that the method *withdraw*$^{(2)}$ does exactly the same thing as

```
If (m(1) is secure relatively to (P(1),Q(1),Inv(1)))
    If (m(2) is secure relatively to(P2',Q2',Inv2'))
        -m(2) is secure relatively to (P(2),Q(2),Inv(2))
    Else
        -m(2) is not secure relatively to (P(2),Q(2),Inv(2))
    EndIf
Else
    -Generate (x0,o0) ∈ E×IC1 / HsecurityIty(1)(x0, o0)=0
    -Generate (x0,o0') ∈ E×IC2 / o0' has the same attributes
    values as o0
     If (P2'(x0,o0')=0)
        - m(2) is not secure relatively to (P(2),Q(2),Inv(2))
     Else
        -Indeterminate Form
     EndIf
EndIf
```

Figure 6. Security test cases for an overriding method.

```
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) && (x1<(bal/InterestRate)))
       this.bal=this.bal-(this.InterestRate)*x1;
       InterestRate = InterestRate/2;}
}
```

Figure 7. *Account*1 and *Account*2 classes.

the method *withdraw*$^{(1)}$ (**Figure 7**). As a result thereof, *withdraw*$^{(2)}$ and *withdraw*$^{(1)}$ are similar on the valid domain *CVD*.

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

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

In order to test the conformity of *withdraw*$^{(1)}$ in class *Account*1, we generate randomly x_1 and the balance values in the interval (–200,200) with N =100 (**Table 2**):

The test result shows that for 100 iterations the constraint $H^{(1)}$ is always true $\left(H^{(1)} = 1\right)$, we can deduce that the *withdraw*$^{(1)}$ method is valid (**Table 2**). In this case it is necessary to test the method *withdraw*$^{(2)}$ relatively to its own constraint $H_{(SI,SO)}$:

- 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 Q_2' must be satisfied.
- The specific invariant Inv_2' must be satisfied.

The constraint Q_2' is always satisfied (**Figures 7** and **8**), however we must proof that Inv_2' is satisfied.

For each created object o_0 ,we have : ($0 \le \text{InterestRate}_0 \le 0.3$) where InterestRate_0 is the initial value assigned to InterestRate when creating the object o_0, and $\text{InterestRate}_{(n)}$ is the value of InterestRate after n operations of type *withdraw*$^{(2)}$ in an execution sequence.

We have:

$$\left[\text{InterestRate}_{(n)} = \frac{\text{InterestRate}_{(n-1)}}{2}\right], n \ge 1$$

Table 2. Result of a conformity test of *withdraw*$^{(1)}$

Iteration number:	*x*	*o*	$P^{(1)}(x,o)$	$H^{(1)}(x,o)$
1	**29**	***Account*1(70)**	**1**	**1**
2	**42**	***Account*1(93)**	**1**	**1**
3	**79**	***Account*1(187)**	**1**	**1**
…	**…**	**…**	**…**	**…**
….	**….**	**….**	**….**	**….**
…..	**…..**	**…..**	**…..**	**…..**
98	**31**	***Account*1(104)**	**1**	**1**
99	**18**	***Account*1(86)**	**1**	**1**
100	**68**	***Account*1(151)**	**1**	**1**

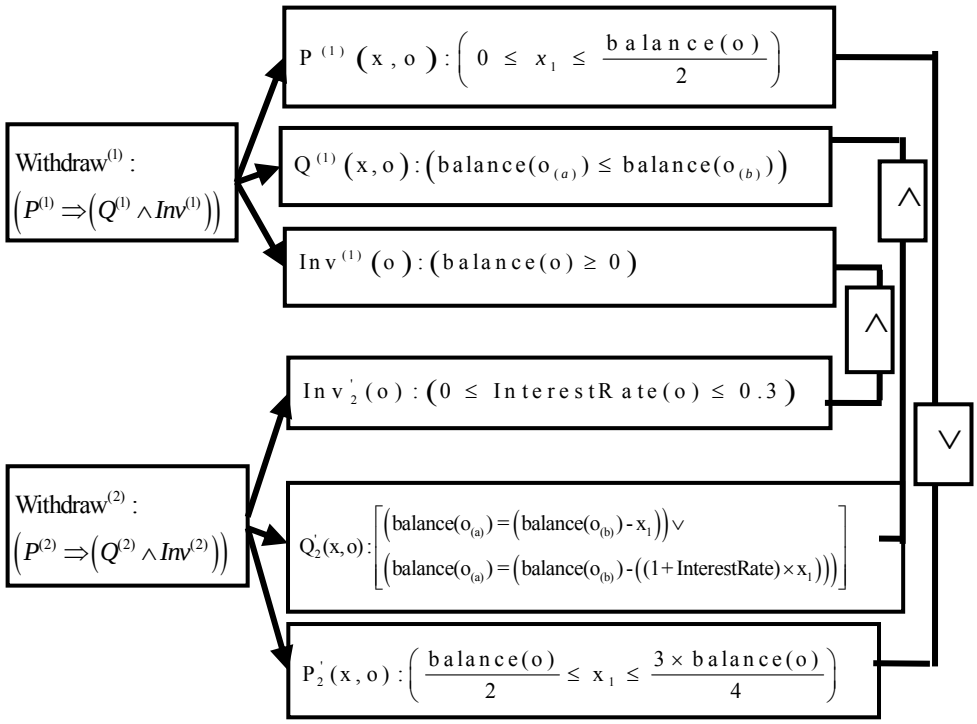

Figure 8. Constraints of *withdraw*$^{(1)}$ and *withdraw*$^{(2)}$.

where n is number of withdrawals (**Figure 7**).

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

$$\left[\text{InterestRate}_{(n)} = \frac{\text{InterestRate}_{(0)}}{2^n}\right],$$

$$\text{with } n \geq 0 \text{ and } \left(0 \leq \text{InterestRate}_0 \geq 0.3\right)$$

We deduce that:

$$\left[\forall n : \left(0 \leq \text{InterestRate}_{(n)} \leq 0.3\right)\right]$$

And consequently, the specific invariant is always satisfied (**Figure 8**). 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.

- Security testing for *withdraw*$^{(2)}$ and *withdraw*$^{(1)}$

For security testing, we test firstly the similarity of withdraw methods on the common invalid domain *CID*

$$\left(CID = \left\{(x,o) \in E \times I_{C_2} / \left(P^{(1)} \vee P_2'\right)(x,o) = 0\right\}\right)$$

For this we generate randomly x_1 and the balance values in the interval (–200,200) with the threshold limit N =100 (**Table 3**).

The test result shows that for 100 iterations the size of the similarity set *Sim* is exactly the threshold limit of the

test. We can conclude that the methods *withdraw*$^{(2)}$ and *withdraw*$^{(1)}$ are similar on the domain *CID* relatively to $\left(P^{(1)}, Q^{(1)}, Inv^{(1)}\right)$ (**Table 3**).

In the last paragraph, we have showed that *withdraw*$^{(2)}$ and *withdraw*$^{(1)}$ are in conformity with their specifications and are similar on the common invalid domain. For testing the security of the overriding method *withdraw*$^{(2)}$, we must testing the security of the overridden method *withdraw*$^{(1)}$ (**Figure 6**):

- Security Testing for *withdraw*$^{(1)}$ relatively to $\left(P^{(1)}, Q^{(1)}, Inv^{(1)}\right)$

We generate in the **Table 4** the security test cases for the overridden method *withdraw*$^{(1)}$ relatively to $\left(P^{(1)}, Q^{(1)}, Inv^{(1)}\right)$:

For the first four iterations, we have:

$$x_1 > \text{balance}(o) > \frac{\text{balance}(o)}{2},$$

i.e. $P^{(1)}(x,o) = 0$ and it induces to a false invariant $(\text{balance}(o) < 0)$ at the output, *i.e.* $H^{(1)}_{securty}(x,o) = 1$. In the iteration 5, we have for

$$(x,o) = (137, Account1(180)):$$

$$\frac{\text{balance}(o)}{2} < x_1 < \text{balance}(o) \ ,$$

Table 3. Similarity test of the *withdraw* methods on *CID*.

Itration number	*x*	*o*	$(P^{(1)} \vee P'_2)(x,o)$	$(x,o) \in$
1	117	*Account*2(96,0.24)	0	*Sim*
2	94	*Account*2(83,0.18)	0	*Sim*
3	173	*Account*2(147,0.01)	0	*Sim*
…	…	…	…	…
….	….	….	…	….
…..	…..	...	…	…..
98	102	*Account*2(120,0.17)	0	*Sim*
99	88	*Account*2(72,0.1)	0	*Sim*
100	131	*Account*2(159,0.22)	0	*Sim*

Table 4. Security test of *withdraw*$^{(1)}$/($P^{(1)}$, $Q^{(1)}$,$Inv^{(1)}$).

Iteration number	x_1	*O*	$P^{(1)}(x,o)$	$H^{(1)}_{security}(x,o)$
1	113	*Account*1(87)	0	1
2	176	*Account*1(138)	0	1
3	91	*Account*1(42)	0	1
4	101	*Account*1(73)	0	1
5	137	*Account*1(180)	0	0

i.e. $P^{(1)}(x,o) = 0$, and this induces $Q^{(1)}(x,o) = 1$ and $Inv^{(1)}(x,o) = 1$, *i.e.* $H^{(1)}_{securty}(x,o) = 0$. Indeed, our implementation cannot reject this situation and conesquently the overridden method *withdraw*$^{(1)}$ under test which is conforming to its specification, is considered not secure relatively to the same specification.

- Security Testing for *withdraw*$^{(2)}$ relatively to $\left(P^{(2)}, Q^{(2)}, Inv^{(2)}\right)$

According to the **Figure 6**, we have for $(x,o) = (137, Account1(180))$ the method *withdraw*$^{(1)}$ is not secure, we consider for example the object *o'* of the class *Account*2 that has the same balance value $(o' = Account2(180, 0.19))$ and we must determinate the truth value of $P'_2(x,o)$ (**Figure 6**).

We have:

$$\left[\left(\frac{1}{2}\right)\times 180 \le 137 \le \left(\frac{3}{4}\right)\times 180 \text{ is false}\right]$$

i.e. $P'_2(x,o) = 0$ (**Figure 8**).

Finally, we can deduce that *withdraw*$^{(2)}$ is not secure relatively to $\left(P^{(2)}, Q^{(2)}, Inv^{(2)}\right)$ (**Figure 6**).

7. Conclusions

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.

The main contribution of our approach is the definition of a security testing that generates invalid input values not satisfying the precondition constraint. Indeed, the test methods have focused only on valid inputs satisfying the pre-condition, and do not integrate the invalid data in test process. We think that a correctly implemented testing method should eliminate cases of invalid data which lead to valid results. Our approach shows how we can use the valid and invalid data extracted from the pre-condition to test the conformity and security of overridden and overriding methods.

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 our approach of security testing based on data not satisfying the precondition constraint.

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, September 19-20, 2005, 2005, pp. 64-71.

[2] F. Bouquet, F. Dadeau, B. Legeard and M. Utting, "Symbolic Animation of JML Specifications," *International Conference on Formal Methods*, Vol. 3582, Springer-Verlag, 2005, 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 International Conference on Network Architectures and Information Systems Security* (*SAR-SSI* 2011), La Rochelle, 18-21 May 2011, pp. 143-150.

[5] K. Benlhachmi and M. Benattou, "A Formal Model of Similarity Testing for Inheritance in Object Oriented Software," *Proceedings of the IEEE International Conference* (CIST'2012), Fez, 24-26 October 2012, 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, No. 11, 2012, 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*, Volume II, 25-28 June 2007, Las Vegas, pp. 385-392.

[8] G. T. Leavens, "JML's Rich, Inherited Specification for Behavioral Subtypes," Iowa State University, Ames, 2006.

[9] R. B. Findler, M. Latendresse and M. Felleisen, "Behavioral Contracts and Behavioral Subtyping," Foundations of Software Engineering, Rice University, Houston, 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, pp. 1811-1841.

[11] G. T. Leavens and K. K. Dhara, "Concepts of Behavioral Subtyping and a Sketch of their Extension to Component-Based Systems," In: G. T. Leavens and M. Sitaraman, Eds., *Foundations of Component-Based Systems*, 2000, pp. 113-135.

[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, Pittsburgh, 1999.

[13] B. Meyer, "Object Oriented Software Construction," Prentice Hall, Upper Saddle River, 1988.

