Test Selection Criteria for Quantifier-Free First-Order Specifications*

Marc Aiguier[1], Agnès Arnould[2], Pascale Le Gall[1], and Delphine Longuet[1]

[1] IBISC CNRS FRE 2873 - University of Évry Val d'Essonne
523 place des terrasses de l'Agora, F-91000 Évry
{marc.aiguier,pascale.legall,delphine.longuet}@ibisc.univ-evry.fr
[2] SIC - University of Poitiers
SP2MI, F-86962 Futuroscope Cedex
arnould@sic.univ-poitiers.fr

Abstract. This paper deals with test case selection from axiomatic specifications whose axioms are quantifier-free first-order formulae. Test cases are modeled as ground formulae and any specification has an exhaustive test data set whose successful submission means correctness, provided that the software under verification can be modeled as a first-order structure over the same signature. As it has already been done for positive conditional equational specifications, we derive test cases from selection criteria based on axiom coverage. Our selection criteria allows us to select test cases by iteratively unfolding an initial target test purpose, given as a formula. The initial reference test set is iteratively split into successive subsets. Each subset of test cases is defined by constraints which are increasingly introduced by the unfolding procedure to ensure an appropriate matching between the current test purpose under unfolding and specification axioms. Our unfolding procedure is sound (no test is added) and complete (no test is lost) with respect to the starting test purpose. It is exemplified on a simple example.

Keywords: Specification-based testing, quantifier-free first-order specifications, selection criteria, test purpose, axiom coverage, unfolding, proof tree normalization.

Introduction

Specification-based testing is a particular case of black-box testing which consists in performing the system under test with some input data in order to state whether its behaviour is conformant to a rigorous specification (*i.e.* given as a formal text provided with a clear semantic). Formal specifications make possible the automation of both test case generation from selection criteria and evaluation

* This work is performed within a French national project STACS (*Spécification et Test, Abstraits et Compositionnels, de Systèmes)* in collaboration with the French Atomic Energy Commission (CEA). This project is devoted to automatically generate test data sets for Input Output Symbolic Transition Systems.

of test executions as successful or not. Selection criteria for specification-based testing generally allow to cover specification requirements (*e.g.* axioms, transitions or states). The computation of the success/failure verdict of test execution tools follows from the comparison between the outputs given by the system under test and the expected ones defined by the formal specification. Besides the possibility of computing verdicts for a test case execution, using formal specifications allows one to properly define the conformance relation, which states what it means for a system to conform to its specification. Such a conformance relation depends on both test hypotheses on the system, which allow to consider it as a formal model, and observability restrictions on the system. These observability restrictions are used to select test cases which can be interpreted as successful or not when performed by the system under test. For instance, in the framework of testing from algebraic specifications, "observable" test cases are any ground equations provided with an equality predicate within the programming language used to implement the system under test. When such conditions (test hypotheses on systems and observability restrictions) are precisely stated, it becomes possible to formally define the testing activity [1, 2]. In particular, correctness can be defined up to these conditions by characterizing an exhaustive test set, whose success is equivalent to system correctness. Moreover, a testing process can be qualified as sound if selected test cases cannot discard correct systems, and as complete if any non-correct system can be detected by at least one test case. In fact, these notions of soundness and completeness may be slightly adapted depending on whether they are applied to an exhaustive test set, to a selection criterion, or to a subset of tests targeted by a test purpose [3].

Testing from algebraic specifications has already been extensively studied [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. Correctness issues have been investigated in presence of non-observable types whose equality can only be observed through observable contexts, *i.e.* by applying some composition of functions yielding an observable result. Selection issues have also been investigated. They consist in either directly covering axioms by instantiating variables with some chosen data or unfolding axioms in order to make a case analysis of function definition. In this last case, test cases for a functionality under test are extracted from the specification by building input data which match the different cases defined by the specification. For example, when functions are recursively specified, the analysis can be refined as many times as the tester chooses to do it. The main drawback of such a selection strategy is that the specification under consideration has to be under a restrictive form, namely positive conditional formulae [4, 5, 6].

In this paper, we propose a family of selection criteria based on axiom unfolding for a larger class of axiomatic specifications: quantifier-free first-order formulae. The enlargement is twofold. First, we do not reduce atomic formulae to equations and consider any kind of predicates. Secondly, formulae are not restricted to Horn clauses (called conditional positive formulae when dealing with equational logic). Our primary goal was to consider the whole classical first-order language. However, we immediately eliminate the existential quantifier. Indeed, testing a formula of the form $\exists X, \varphi(X)$ would amount to exhibit a witness value

a such that $\varphi(X)$ is interpreted as true by the system when substituting X by a. Of course, there is no general way to exhibit such a pertinent value, but notice that astonishly, exhibiting such a value would amount to simply prove the system with respect to the initial property. Thus, existential properties are not testable. Some works on specification-based testing [7,8] have already considered a similar class of formulae. They propose a mixed approach combining black-box and white-box testing to deal with the problem of non-observable data types. From the selection point of view, they do not propose any particular strategy, but only the substitution of axiom variables for some arbitrarily chosen data. On the contrary, following the specification-based testing framework proposed in [1], we characterize an exhaustive test set for such specifications. Moreover, by extending the unfolding-based selection criteria family defined for conditional positive equational specifications, we define a sound and complete unfolding procedure devoted to the coverage of quantifier-free first-order axioms.

The paper is organized as follows. In Section 1, we recall standard notations about quantifier-free first-order specifications. Section 2 gives relevant definitions of [1] concerning our framework of testing. In Section 3, an exhaustive test set for quantifier-free first-order specifications is characterized. Section 4 proposes an unfolding procedure allowing us to define a family of selection criteria for the considered class of specifications. Finally, in Section 4.3, the selection criteria based on the unfolding procedure is proved to be both sound and complete.

1 Preliminaries

1.1 Quantifier-Free First-Order Specifications

A *(first-order) signature* $\Sigma = (S, F, P, V)$ consists of a set S of sorts, a set F of operation names each one equipped with an arity in $S^* \times S$, a set P of predicate names each one equipped with an arity in S^+ and an S-indexed set of variables V. In the sequel, an operation name f of arity $(s_1 \dots s_n, s)$ will be denoted by $f : s_1 \times \dots \times s_n \to s$, and a predicate name p of arity $(s_1 \dots s_n)$ will be denoted by $p : s_1 \times \dots \times s_n$. Given a signature $\Sigma = (S, F, P, V)$, $T_\Sigma(V)$ and T_Σ are both S-sets of *terms with variables in* V and *ground terms*, respectively, freely generated from variables and operations in Σ and preserving arity of operations. A *substitution* is any mapping $\sigma : V \to T_\Sigma(V)$ that preserves sorts. Substitutions are naturally extended to terms with variables. *Σ-atomic formulae* are formulae of the form $p(t_1, \dots, t_n)$ with $p : s_1 \times \dots \times s_n$ and $t_i \in T_\Sigma(V)_{s_i}$ for each i, $1 \leq i \leq n$. A *Σ-formula* is a quantifier-free first-order formula built from atomic formulae and Boolean connectives $\neg$, $\wedge$, $\vee$ and $\Rightarrow$. As usual, free variables of quantifier-free formulae are implicitly universally quantified. A Σ-formula is said *ground* if it does not contain variables. Let us denote $For(\Sigma)$ the set of all Σ-formulae. A *specification* $Sp = (\Sigma, Ax)$ consists of a signature Σ and a set Ax of quantifier-free formulae built over Σ. Formulae in Ax are often called *axioms*.

A *Σ-model* $\mathcal{M}$ is an S-indexed set M equipped for each $f : s_1 \times \dots \times s_n \to s \in F$ with a mapping $f^\mathcal{M} : M_{s_1} \times \dots \times M_{s_n} \to M_s$ and for each predicate $p : s_1 \times \dots \times s_n$ with an n-ary relation $p^\mathcal{M} \subseteq M_{s_1} \times \dots \times M_{s_n}$. $Mod(\Sigma)$ is the category objects of which are all Σ-models. Given a Σ-model $\mathcal{M}$, a Σ-interpretation in M is any mapping $\nu : V \to M$. Interpretations are naturally extended to terms with variables. A Σ-model $\mathcal{M}$ *satisfies* for an interpretation ν a Σ-atomic formula $p(t_1, \ldots, t_n)$ if and only if $(\nu(t_1), \ldots, \nu(t_n)) \in p^{\mathcal{M}}$. The *satisfaction* of a Σ-formula φ for an interpretation ν by $\mathcal{M}$, denoted $\mathcal{M} \models_\nu \varphi$, is inductively defined on the structure of φ from the satisfaction for ν of atomic formulae of φ and using classic semantic interpretations of Boolean connectives. $\mathcal{M}$ validates a formula φ, denoted $\mathcal{M} \models \varphi$, if and only if for every interpretation $\nu : V \to M$, $\mathcal{M} \models_\nu \varphi$. Given $\Psi \subseteq For(\Sigma)$ and two Σ-models $\mathcal{M}$ and $\mathcal{M}'$, $\mathcal{M}$ is *Ψ-equivalent to $\mathcal{M}'$*, denoted $\mathcal{M} \equiv_\Psi \mathcal{M}'$, if and only if we have: $\forall \varphi \in \Psi,\ \mathcal{M} \models \varphi \iff \mathcal{M}' \models \varphi$. Given a specification $Sp = (\Sigma, Ax)$, a Σ-model $\mathcal{M}$ is an *Sp-model* if for every $\varphi \in Ax$, $\mathcal{M} \models \varphi$. $Mod(Sp)$ is the full subcategory of $Mod(\Sigma)$, objects of which are all Sp-models. A Σ-formula φ is a *semantic consequence* of a specification $Sp = (\Sigma, Ax)$, denoted $Sp \models \varphi$, if and only if for every Sp-model $\mathcal{M}$, we have $\mathcal{M} \models \varphi$. $Sp^\bullet$ is the set of all semantic consequences.

Given a set of quantifier-free formulae $\Psi \subseteq For(\Sigma)$, let us denote $\mathcal{H}_{T_\Sigma}$ the Σ-model, classically called the Herbrand model of Ψ,

– defined by the Σ-algebra, whose carrier is T_Σ and whose operation meaning is defined for every operation $f : s_1 \times \ldots \times s_n \to s \in F$ by the mapping $f^{\mathcal{H}_{T_\Sigma}} : (t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)$, and
– determined by the set of ground atomic formulae $p(t_1, \ldots, t_n)$ such that $\Psi \models p(t_1, \ldots, t_n)$.

It is easy to show that $\Psi \models \varphi \Leftrightarrow \mathcal{H}_{T_\Sigma} \models \varphi$ for every ground formula φ, and then $\mathcal{H}_{T_\Sigma} \in Mod((\Sigma, \Psi))$.

A calculus for quantifier-free first-order specifications is defined by the following inference rules, where $\Gamma \mathrel{|\!\sim} \Delta$ is a sequent such that Γ and Δ are two sets of quantifier-free first-order formulae:

$$\frac{}{\Gamma,\varphi \mathrel{|\!\sim} \Delta,\varphi}\ \mathsf{Ax}$$

$$\frac{\Gamma \mathrel{|\!\sim} \Delta,\varphi}{\Gamma,\neg\varphi \mathrel{|\!\sim} \Delta}\ \mathsf{Left}\text{-}\neg \qquad \frac{\Gamma,\varphi \mathrel{|\!\sim} \Delta}{\Gamma \mathrel{|\!\sim} \Delta,\neg\varphi}\ \mathsf{Right}\text{-}\neg$$

$$\frac{\Gamma,\varphi,\psi \mathrel{|\!\sim} \Delta}{\Gamma,\varphi\wedge\psi \mathrel{|\!\sim} \Delta}\ \mathsf{Left}\text{-}\wedge \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta,\varphi \quad \Gamma \mathrel{|\!\sim} \Delta,\psi}{\Gamma \mathrel{|\!\sim} \Delta,\varphi\wedge\psi}\ \mathsf{Right}\text{-}\wedge$$

$$\frac{\Gamma,\varphi \mathrel{|\!\sim} \Delta \quad \Gamma,\psi \mathrel{|\!\sim} \Delta}{\Gamma,\varphi\vee\psi \mathrel{|\!\sim} \Delta}\ \mathsf{Left}\text{-}\vee \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta,\varphi,\psi}{\Gamma \mathrel{|\!\sim} \Delta,\varphi\vee\psi}\ \mathsf{Right}\text{-}\vee$$

$$\frac{\Gamma \mathrel{|\!\sim} \Delta,\varphi \quad \Gamma,\psi \mathrel{|\!\sim} \Delta}{\Gamma,\varphi\Rightarrow\psi \mathrel{|\!\sim} \Delta}\ \mathsf{Left}\text{-}\Rightarrow \qquad \frac{\Gamma,\varphi \mathrel{|\!\sim} \Delta,\psi}{\Gamma \mathrel{|\!\sim} \Delta,\varphi\Rightarrow\psi}\ \mathsf{Right}\text{-}\Rightarrow$$

$$\frac{\Gamma \mathrel{|\!\sim} \Delta}{\sigma(\Gamma) \mathrel{|\!\sim} \sigma(\Delta)}\ \mathsf{Subs} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta,\varphi \qquad \Gamma',\varphi \mathrel{|\!\sim} \Delta'}{\Gamma,\Gamma' \mathrel{|\!\sim} \Delta,\Delta'}\ \mathsf{Cut}$$

Observe that the inference rules associated to Boolean connectives obviously define an automatic process that allows to transform any sequent $\mathrel{|\!\sim} \varphi$, where φ is a quantifier-free formula, into a set of sequents $\Gamma \mathrel{|\!\sim} \Delta$ where every formula in Γ and Δ is atomic. Let us call such sequents *normalized sequents*.

Moreover, we can show that every proof tree can be transformed into a proof tree of same conclusion and such that both Cut and Subs rules never occur under rule instances associated to Boolean connectives. This transformation is obtained from basic transformations, for example:

$$\dfrac{\dfrac{\Gamma \mathrel{|\!\!\sim} \Delta,\psi,\varphi}{\Gamma,\neg\varphi \mathrel{|\!\!\sim} \Delta,\psi}\,\textsf{Left-}\neg \qquad \Gamma',\psi \mathrel{|\!\!\sim} \Delta'}{\Gamma,\Gamma',\neg\varphi \mathrel{|\!\!\sim} \Delta,\Delta'}\,\textsf{Cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \mathrel{|\!\!\sim} \Delta,\psi,\varphi \qquad \Gamma',\psi \mathrel{|\!\!\sim} \Delta'}{\Gamma,\Gamma' \mathrel{|\!\!\sim} \Delta,\Delta',\varphi}\,\textsf{Cut}}{\Gamma,\Gamma',\neg\varphi \mathrel{|\!\!\sim} \Delta,\Delta'}\,\textsf{Left-}\neg$$

The other basic transformations are defined in the same way. Therefore, using proof terms for proofs, with a recursive path ordering $>^{rpo}$ to order proofs induced by the well-founded relation (precedence) $>$ on rule instances

$$\textsf{Cut, Subs} > \textsf{Left-@, Right-@}, \text{ where } @ \in \{\neg, \wedge, \vee, \Rightarrow\}$$

we show that the transitive closure of $\rightsquigarrow$is contained in the relation $>^{rpo}$, and thus that $\rightsquigarrow$ is terminating.

This last result states that every sequent is equivalent to a normalized sequent, which allows to only deal with normalized sequents. Therefore, in the following, we will suppose that specification axioms are normalized sequents.

1.2 Running Example

By way of illustration, we give a specification of sorted lists of positive rationals.

We first give a specification of naturals, built from constructors 0 and successor s. Addition *add* and multiplication *mult* on naturals are specified as usual, as well as the predicate "less than" *ltn*. The constructor operation $\_/\_$ then builds rationals from couples of naturals. Two rationals x/y and u/v are equal (*eqr* predicate) if $mult(x, v)$ and $mult(u, y)$ are equal. Since we consider only positive rationals, x/y is less than u/v (*ltr* predicate) if $mult(x, v)$ is less than $mult(u, y)$.

Lists of rationals are then built from constructors $[\,]$ and $\_ :: \_$ as usual. The insertion *insert* of a rational in a sorted list needs to consider four cases: the list is empty; the first element of the list is equal to the rational to insert, and then the element is not repeated; the first element of the list is greater than the rational to insert, and then it is inserted at the head; the first element of the list is less than the rational to insert, then the insertion is tried in the rest of the list. The membership predicate *isin* is specified saying that there is no element in the empty list, and that searching for an element in a non-empty list comes to find it at the head of the list or to search it in the rest of the list.

The behaviour of operations *add*, *mult* and *insert* is classically specified by equations. When dealing with first-order logic, this requires to introduce three equality predicates $=_{Nat}$: $Nat \times Nat$, $=_{Rat}$: $Rat \times Rat$ and $=_{List}$: $List \times List$, each one equipped with the following axioms:

$$\begin{array}{l}
x =_@ x \\
x =_@ y \Rightarrow y =_@ x \\
x =_@ y \wedge y =_@ z \Rightarrow x =_@ z \\
x_1 =_{@_1} y_1 \wedge \ldots \wedge x_n =_{@_n} y_n \Rightarrow f(x_1, \ldots, x_n) =_@ f(y_1, \ldots, y_n) \\
x_1 =_{@_1} y_1 \wedge \ldots \wedge x_n =_{@_n} y_n \wedge p(x_1, \ldots, x_n) \Rightarrow p(y_1, \ldots, y_n)
\end{array}$$

where $@, @_i \in \{Nat, Rat, List\}$, $f : @_1 \times \ldots \times @_n \to @$ and $p : @_1 \times \ldots \times @_n$. In order not to make heavy specifications, another approach is to transform any operation $f : s_1 \times \ldots \times s_n \to s$ into a predicate $f : s_1 \times \ldots \times s_n \times s$ and then to make the equality implicit. This is the approach we will follow in the specification below. Another consequence of such an approach is to make the use of our algorithm of selection criteria, based on axiom unfolding, easier because less axioms are considered.

spec RatList =
  sorts *Nat*, *Rat*, *List*
  ops $0 : Nat$;
    $s : Nat \to Nat$;
    $\_\_/\_\_ : Nat \times Nat \to Rat$;
    $[\,] : List$;
    $\_\_::\_\_ : Rat \times List \to List$
  preds $add : Nat \times Nat \times Nat$;
    $mult : Nat \times Nat \times Nat$;
    $ltn : Nat \times Nat$;
    $eqr : Rat \times Rat$;
    $ltr : Rat \times Rat$;
    $insert : Rat \times List \times List$;
    $isin : Rat \times List$
  vars x, y, z, u, v, n, m: *Nat*; e: *Rat*; l, l': *List*
  • $add(x, 0, x)$
  • $add(x, s(y), s(z)) \Leftrightarrow add(x, y, z)$
  • $mult(x, 0, 0)$
  • $add(x, u, z) \wedge mult(x, y, u) \Rightarrow mult(x, s(y), z)$
  • $ltn(0, s(x))$
  • $\neg\, ltn(x, 0)$
  • $ltn(s(x), s(y)) \Leftrightarrow ltn(x, y)$
  • $mult(x, s(v), n) \wedge mult(u, s(y), n) \Rightarrow eqr(x/s(y), u/s(v))$
  • $ltn(m, n) \wedge mult(x, s(v), m) \wedge mult(u, s(y), n) \Rightarrow ltr(x/s(y), u/s(v))$
  • $insert(x/s(y), [\,], x/s(y) :: [\,])$
  • $eqr(x/s(y), e) \Rightarrow insert(x/s(y), e :: l, e :: l)$
  • $ltr(x/s(y), e) \Rightarrow insert(x/s(y), e :: l, x/s(y) :: (e :: l))$
  • $ltr(e, x/s(y)) \wedge insert(x/s(y), l, l') \Rightarrow insert(x/s(y), e :: l, e :: l')$
  • $\neg\, isin(x/s(y), [\,])$
  • $isin(x/s(y), e :: l) \Leftrightarrow eqr(x/s(y), e) \vee isin(x/s(y), l)$
end

Axioms are then transformed into normalized sequents, as explained above. For example, the normalization of the right-to-left implication of the axiom $isin(x/s(y), e :: l) \Leftrightarrow eqr(x/s(y), e) \vee isin(x/s(y), l)$ leads to two normalized sequents as follows:

$$\dfrac{\dfrac{eqr(x/s(y),e) \mathrel{|\!\!\sim} isin(x/s(y),e::l) \qquad isin(x/s(y),l) \mathrel{|\!\!\sim} isin(x/s(y),e::l)}{eqr(x/s(y),e) \vee isin(x/s(y),l) \mathrel{|\!\!\sim} isin(x/s(y),e::l)}\,\text{Left-}\vee}{\mathrel{|\!\!\sim} eqr(x/s(y),e) \vee isin(x/s(y),l) \Rightarrow isin(x/s(y),e::l)}\,\text{Right-}\Rightarrow$$

1. $\mid\!\sim add(x, 0, x)$
2. $add(x, s(y), s(z)) \mid\!\sim add(x, y, z)$
3. $add(x, y, z) \mid\!\sim add(x, s(y), s(z))$
4. $\mid\!\sim mult(x, 0, 0)$
5. $add(x, u, z), mult(x, y, u) \mid\!\sim mult(x, s(y), z)$
6. $\mid\!\sim ltn(0, s(x))$
7. $ltn(x, 0) \mid\!\sim$
8. $ltn(s(x), s(y)) \mid\!\sim ltn(x, y)$
9. $ltn(x, y) \mid\!\sim ltn(s(x), s(y))$
10. $mult(x, s(v), n), mult(u, s(y), n) \mid\!\sim eqr(x/s(y), u/s(v))$
11. $ltn(m, n), mult(x, s(v), m), mult(u, s(y), n) \mid\!\sim ltr(x/s(y), u/s(v))$
12. $\mid\!\sim insert(x/s(y), [\,], x/s(y) :: [\,])$
13. $eqr(x/s(y), e) \mid\!\sim insert(x/s(y), e :: l, e :: l)$
14. $ltr(x/s(y), e) \mid\!\sim insert(x/s(y), e :: l, x/s(y) :: e :: l)$
15. $ltr(e, x/s(y)), insert(x/s(y), l, l') \mid\!\sim insert(x/s(y), e :: l, e :: l')$
16. $isin(x/s(y), [\,]) \mid\!\sim$
17. $isin(x/s(y), e :: l) \mid\!\sim eqr(x/s(y), e), isin(x/s(y), l)$
18. $eqr(x/s(y), e) \mid\!\sim isin(x/s(y), e :: l)$
19. $isin(x/s(y), l) \mid\!\sim isin(x/s(y), e :: l)$

2 A General Framework of Testing from Formal Specifications

The work presented in Section 4 comes within the general framework of testing from formal specifications defined in [1]. Here, we succinctly introduce this framework, then we instantiate it to the formalism we have just defined in Section 1.

The interpretation of test cases submission as a success or failure is related to the notion of program correctness. Following previous works [1, 4, 9, 10, 11], test cases are formulae and programs are Σ-models. Therefore, test cases interpretation is defined by formula satisfaction. When a test case is submitted to a program, it has to yield a verdict (success or failure). Hence, test cases have to be directly interpreted as "true" or "false" by a "computation" of the program. These "executable" formulae are called *observable*.

Let $Sp = (\Sigma, Ax)$ be a specification and $Obs \subseteq For(\Sigma)$ any set of observable formulae. Let P be a program which is denoted by a Σ-model of $Mod(\Sigma)$. Then *test cases* are observable formulae, which are *successful* for P if and only if P validates them (i.e. performs them and interprets them as "true"). A *test set* T is then a set of test cases. T is said to be *successful* for P if and only if $\forall \varphi \in T, P \models \varphi$.

Following an observational approach [14], to be qualified as correct with respect to a specification Sp, a program is required to be observationally equivalent to a model of $Mod(Sp)$, up to the observable formulae of Obs.

Definition 1 (Correctness). *P is* correct *for Sp via Obs, denoted by $Correct_{Obs}(P, Sp)$, if and only if there exists a model $\mathcal{M}$ in $Mod(Sp)$ such that $\mathcal{M} \equiv_{Obs} P$.*

Definition 2 (Exhaustiveness). *Let $\mathcal{K} \subseteq Mod(\Sigma)$. A test set T is* exhaustive *for $\mathcal{K}$ with respect to Sp and Obs if and only if*

$$\forall P \in \mathcal{K}, P \models T \Longleftrightarrow Correct_{Obs}(P, Sp)$$

The existence of an exhaustive test set means that Sp is testable via Obs since correctness can be asymptotically approached by submitting a (possibly infinite) test set. Hence, an exhaustive test set is appropriate to start the process of selecting a finite test set with a reasonable size. However, depending on the nature of Sp, Obs and $\mathcal{K}$, an exhaustive test set does not necessarily exist. For instance, in [12], we have shown that for positive conditional algebraic specifications, when Obs is restricted to ground equations, $Sp^\bullet \cap Obs$ is only exhaustive for algebras satisfying a strong condition, called initiality, which, roughly speaking, means that the program under test behaves like the initial algebra of $Mod(Sp)$ for all ground instances of equations occurring in premises of axioms of Sp. The problem is that showing such a property on a program may be as difficult as proving its correctness, and then restricts its testability.

In Section 3, we will show that in the presence of a specification Sp with quantifier-free axioms, and when the set of observable formulae Obs is the set of all ground first-order formulae, the exhaustiveness of $Sp^\bullet \cap Obs$ holds without conditions on programs, that is $\mathcal{K} = Mod(\Sigma)$.

Test sets can be compared with respect to their ability to reject (or to accept, from a dual point of view) programs. Two test sets are then said to be equivalent if and only they accept exactly the same programs.

The challenge of testing then consists in managing (infinite) test sets. In practice, experts apply some selection criteria on a reference test set in order to extract a test set of sufficiently reasonable size to be submitted to the program. The underlying idea is that all test sets satisfying a considered selection criterion reveal the same class of incorrect programs, intuitively those corresponding to the fault model captured by the criterion. For example, the criterion called "uniformity hypothesis" over a test set T postulates that any chosen value is equivalent to another one in T.

A classic way to select test data with a selection criterion C consists in splitting a given starting test set T into a family of test subsets $\{T_i\}_{i \in I_{C(T)}}$ such that $T = \cup_{i \in I_{C(T)}} T_i$ holds. A test set satisfying such a selection criterion simply contains at least one test case for each non-empty subset T_i. Intuitively, all test cases in T_i are supposed equivalent to reveal incorrect programs with respect to the fault model captured by T_i. Hence, the selection criterion C is a coverage criterion according to the way C is splitting the initial test set T into the family $\{T_i\}_{i \in I_{C(T)}}$. This is the method that we will use in this paper to select test data, known under the term of *partition testing*.

For instance, the selection criterion we will define in the sequel of this paper consists in splitting a test set into subsets according to specification axioms. If we come back to the RatList specification, the *insert* predicate is specified inductively by four axioms. Testing a formula consists in finding input data, that is, ground substitutions to apply to the formula in order to submit it to the program, bringing into play at least once each of these four axioms. Therefore, the set of test cases associated to $insert(r, L, L')$, where r, L and L' are variables, can be split into four subsets:

1. The set of tests associated to the substitution $L \mapsto [\,]$, coming from the axiom $insert(x/s(y), [\,], x/s(y) :: [\,])$.
2. The set associated to the case where the rational to insert is equal to the first element of the list, that is, associated to the substitution $r \mapsto x/s(y)$, $L \mapsto e :: l$ with $eqr(x/s(y), e)$, coming from the axiom $eqr(x/s(y), e) \Rightarrow insert(x/s(y), e :: l, e :: l)$.
3. The set associated to the case where it is less than the first element, that is, the substitution $r \mapsto x/s(y)$, $L \mapsto e :: l$ with $ltr(x/s(y), e)$, coming from the axiom $ltr(x/s(y), e) \Rightarrow insert(x/s(y), e :: l, x/s(y) :: e :: l)$.
4. The set associated to the case where it is greater than it, that is, the substitution $r \mapsto x/s(y)$, $L \mapsto e :: l$ with $ltr(x/s(y), e)$, coming from the axiom $ltr(e, x/s(y)) \wedge insert(x/s(y), l, l') \Rightarrow insert(x/s(y), e :: l, e :: l')$.

The process can be pursued on each above subset.

Definition 3 (Selection criterion). *A* selection criterion *C is a mapping*[1] $\mathcal{P}(Sp^\bullet \cap Obs) \to \mathcal{P}(\mathcal{P}(Sp^\bullet \cap Obs))$. *For a test set T, we denote $|C(T)| = \cup_{i \in I_{C(T)}} T_i$ where $C(T) = \{T_i\}_{i \in I_{C(T)}}$.*

T' satisfies C applied to T*, denoted by $T' \sqsubset C(T)$, if and only if:*

$$\forall i \in I_{C(T)}, T_i \neq \emptyset \Rightarrow T' \cap T_i \neq \emptyset$$

A selection criterion consists of a mapping that splits test sets into families of test sets. The selection criterion is satisfied as soon as the considered test set contains at least one test case within each (non-empty) set of the resulting family. To be pertinent, a selection criterion should ensure some properties between the starting test set and the resulting family of test sets:

Definition 4 (Properties). *Let C be a selection criterion and T be a test set.*

– *C is said* sound for *T if and only if $|C(T)| \subseteq T$;*
– *C is said* complete for *T if and only if $|C(T)| = T$.*

The properties of soundness and completeness are essential for an adequate selection criterion: soundness ensures that test cases will be selected within the starting test set (i.e. no test is added) while completeness ensures that we capture all test cases up to the notion of equivalent test cases (i.e. no test is lost).

3 An Exhaustive Test Set

Here, we show that for every quantifier-free first-order specification $Sp = (\Sigma, Ax)$, $Sp^\bullet \cap Obs$ is an exhaustive test set for $Mod(\Sigma)$, when Obs is the set of all ground formulae built over Σ.

[1] For a given set X, $\mathcal{P}(X)$ denotes the set of all subsets of X.

Theorem 1. *Let $Sp = (\Sigma, Ax)$ be a specification. Then $Sp^\bullet \cap Obs$ is exhaustive for $Mod(\Sigma)$.*

Proof. Let P be a program, *i.e.* $P \in Mod(\Sigma)$, such that $P \models Sp^\bullet \cap Obs$. Let us show that $Correct_{Obs}(P, Sp)$.

Note $Th(P) = \{\varphi \in Obs \mid P \models \varphi\}$. Let $\mathcal{H}_{T_\Sigma} \in Mod(\Sigma)$ be the Herbrand model of $Th(P)$. By definition, we have that $P \equiv_{Obs} \mathcal{H}_{T_\Sigma}$. Let us then show that $\mathcal{H}_{T_\Sigma} \in Mod(Sp)$. Let φ be an axiom of Sp. Let $\nu : V \to \mathcal{H}_{T_\Sigma}$ be an interpretation. By definition, $\nu(\varphi)$ is a ground formula. By hypothesis, $P \models \nu(\varphi)$ and then $\mathcal{H}_{T_\Sigma} \models \nu(\varphi)$. We conclude that $\mathcal{H}_{T_\Sigma} \models_\nu \varphi$.

Suppose that there exists $\mathcal{M} \in Mod(Sp)$ such that $\mathcal{M} \equiv_{Obs} P$. Let $\varphi \in Sp^\bullet \cap Obs$. By hypothesis, $\mathcal{M} \models \varphi$, then $P \models \varphi$ as well. □

4 Selection Criteria Based on Axiom Unfolding

In this section, we study the problem of test case selection for quantifier-free specifications, by adapting a selection criteria based on unfolding of positive conditional formulae in the algebraic specification setting [6].

4.1 Test Sets for Quantifier-Free Formulae

The selection method that we are going to define takes inspiration from classic methods that split the initial test set of any formula considered as a test purpose. Succinctly, for a quantifier-free first-order formula φ, our method consists in

1. splitting the initial test set for φ into many test subsets, called *constrained test sets for φ*, and
2. choosing any input in each non-empty subset.

First, let us define what test set and constrained test set for a quantifier-free formula are.

Definition 5 (Test set). *Let φ be a quantifier-free formula, called* test purpose. *The* test set for *φ, denoted by T_φ, is the set defined as follows:*

$$T_\varphi = \{\rho(\varphi) \mid \rho : V \to T_\Sigma, \rho(\varphi) \in Sp^\bullet \cap Obs\}$$

Note that φ may be any formula, not necessarily in $Sp^\bullet$.

Example 1. Here are some test purposes for the signature of specification RatList, with examples of associated test cases.

add$(x, 0, x)$**.** Since *add*$(x, 0, x)$ is an axiom, all ground instances of this formula are test cases: *add*$(0, 0, 0)$, *add*$(6, 0, 6)$, *etc.*

eqr(u, v)**.** This predicate is under-specified, the case where a rational is of the form $x/0$ is not taken into account, so there cannot be tests on this case. Test cases may be: *eqr*$(1/2, 1/2)$, *eqr*$(3/6, 4/8)$, *etc.*

$add(m,n,r) \Rightarrow mult(m,2,r)$. Only cases where $add(m,n,r)$ is not satisfied or where $m = n$ are semantic consequences of the specification. The interesting test cases are those where $m = n$ such as $add(2,2,4) \Rightarrow mult(2,2,4)$, $add(5,5,10) \Rightarrow mult(5,2,10)$, *etc.*

$insert(r,l,[\,])$. The formula is never satisfied for any ground instance of r and l, so there is no possible test case.

Definition 6 (Constrained test set). *Let φ be a quantifier-free formula, $\mathcal{C}$ be a set of quantifier-free formulae called Σ-constraints, and $\sigma : V \to T_\Sigma(V)$ be a substitution. A* test set for φ with respect to $\mathcal{C}$ and σ, *denoted by $T_{(\mathcal{C},\sigma),\varphi}$, is the set of ground formulae defined by:*

$$T_{(\mathcal{C},\sigma),\varphi} = \{\rho(\sigma(\varphi)) \mid \rho : V \to T_\Sigma, \rho(\sigma(\varphi)) \in Sp^\bullet \cap Obs, \forall \psi \in \mathcal{C}, \rho(\psi) \in Sp^\bullet \cap Obs\}$$

The couple $\langle(\mathcal{C},\sigma),\varphi\rangle$ is called a constrained test purpose.

Note that the test purpose φ of Definition 5 can be seen as the constrained test purpose $\langle(\{\varphi\}, id),\varphi\rangle$.

Example 2. Let us denote a substitution $\sigma : V \to T_\Sigma(V)$ mapping a set $X = \{x_1,\ldots,x_n\}$ to a set $Y = \{y_1,\ldots,y_n\}$, such that $\sigma(x_i) = y_i$ for all i, $1 \le i \le n$, by $[x_1 \mapsto y_1,\ldots,x_n \mapsto y_n]$.

Examples of constrained test purposes may be the following:

$$\langle(\emptyset,[x \mapsto s(u)]), add(x,0,x)\rangle$$

$$\langle(\{ltn(3,x)\}, id), add(x,0,x)\rangle$$

$$\langle(\{ltn(x,z)\},[u \mapsto x/s(y), v \mapsto z/s(y)]), ltr(u,v)\rangle$$

$$\langle(\{ltn(m,n), mult(x,s(z),m), mult(w,s(y),n)\},[u \mapsto x/s(y), v \mapsto w/s(z)]), ltr(u,v)\rangle$$

As another example, to come back to the example of splitting the test set associated to $insert(r,L,L')$ into four subsets, we can express each of four test subsets in terms of constrained test purposes as follows:

$$\langle(\emptyset,\sigma_1), insert(r,L,L')\rangle$$
$$\langle(\{eqr(x_0/s(y_0),e_0)\},\sigma_2), insert(r,L,L')\rangle$$
$$\langle(\{ltr(x_0/s(y_0),e_0)\},\sigma_3), insert(r,L,L')\rangle$$
$$\langle(\{ltr(e_0,x_0/s(y_0)), insert(x_0/s(y_0),l_0,l'_0)\},\sigma_4), insert(r,L,L')\rangle$$

where

	r	L	L'
σ_1	$x_0/s(y_0)$	$[\,]$	$x_0/s(y_0) :: [\,]$
σ_2	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l_0$
σ_3	$x_0/s(y_0)$	$e_0 :: l_0$	$x_0/s(y_0) :: (e_0 :: l_0)$
σ_4	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l'_0$

Only this kind of constrained test sets, built from a case analysis of the specification axioms, will be of interest. The aim of the unfolding procedure we will introduce in the next section is to build such test sets.

4.2 Unfolding Procedure

In practice, the initial test purpose is unconstrained. The aim is to replace it with a set of constrained test purposes. This is what the unfolding procedure does, matching the initial formula with the specification axioms, when it is possible.

Therefore, the unfolding procedure inputs are:

- a quantifier-free specification $Sp = (\Sigma, Ax)$ where axioms of Ax have been transformed into normalized sequents;
- a quantifier-free formula φ seen as the initial constrained test purpose $\langle(\emptyset, id), \varphi\rangle$;
- a family Ψ of couples $(\mathcal{C}, \sigma)$ where $\mathcal{C}$ is a set of Σ-constraints in the form of normalized sequents, and σ is a substitution $V \to T_\Sigma(V)$.

The first set Ψ_0 only contains the couple composed of the set of normalized sequents obtained from the quantifier-free formula φ under test and the identity substitution.

The unfolding procedure is expressed by the following two rules:[2]

$$\textbf{Reduce}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\Gamma \mathrel{|\!\sim} \Delta\}, \sigma')\}}{\Psi \cup \{(\sigma(\mathcal{C}), \sigma \circ \sigma')\}} \quad \exists\gamma \in \Gamma, \exists\delta \in \Delta \text{ s.t. } \sigma(\gamma) = \sigma(\delta),\ \sigma \text{ mgu}$$

$$\textbf{Unfold}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\psi\}, \sigma')\}}{\Psi \cup \displaystyle\bigcup_{(c,\sigma) \in Tr(\psi)} \{(\sigma(\mathcal{C}) \cup c, \sigma \circ \sigma')\}}$$

where $Tr(\psi)$ for $\psi = \gamma_1, \ldots, \gamma_m \mathrel{|\!\sim} \delta_1, \ldots, \delta_n$ is the set defined by:

$$\left\{ \left(\begin{array}{l} \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m), \sigma(\zeta_i) \mathrel{|\!\sim} \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1 \le i \le k} \\ \cup\ \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m) \mathrel{|\!\sim} \sigma(\xi_i), \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1 \le i \le l} \end{array}, \sigma \right) \middle| \begin{array}{l} \psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mathrel{|\!\sim} \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q \in Ax, \\ 1 \le p \le m, \forall 1 \le i \le p, \sigma(\psi_i) = \sigma(\gamma_i), \\ 1 \le q \le n, \forall 1 \le i \le q, \sigma(\varphi_i) = \sigma(\delta_i), \\ \sigma \text{ unifier}, k, l \in \mathbb{N} \end{array} \right\}$$

The **Red** rule eliminates tautologies from constraints sets. Intuitively, the **Unfold** rule consists in replacing the formula ψ with a set c of constraints, which are what remains of the axiom after unification. Then testing $\sigma(\psi)$ comes to test the formulae of c. The particular case where no formula has to be cut is taken into account, since k and l may be equal to zero. $Tr(\psi)$ is then a couple $(\emptyset, \sigma)$, and it is the last step of unfolding for this formula.

Each unification with an axiom leads to a couple (c, σ), so the initial formula ψ is replaced with as much sets of formulae as there are axioms to which it can be unified. The definition of $Tr(\psi)$ being based on unification, this set is computable if the specification Sp has a finite set of axioms. Therefore, given an

[2] The most general unifier (or mgu) of two terms γ and δ is the most general substitution σ such that $\sigma(\gamma) = \sigma(\delta)$.

atomic formula ψ, we have the selection criterion C_ψ that maps any $T_{(\mathcal{C},\sigma'),\varphi}$ to $(T_{(\sigma(\mathcal{C}\setminus\{\psi\})\cup c,\sigma\circ\sigma'),\varphi})_{(c,\sigma)\in Tr(\psi)}$ if $\psi \in \mathcal{C}$, and to $T_{\mathcal{C},\varphi}$ otherwise.

We write $\langle \Psi, \varphi\rangle \vdash_U \langle \Psi', \varphi\rangle$ to mean that Ψ' can be derived from Ψ by applying **Reduce** or **Unfold**. An unfolding procedure is then a program, inputs of which are a quantifier-free first-order specification *Sp* and a quantifier-free formula φ, and uses the above inference rules to generate the sequence

$$\langle \Psi_0, \varphi\rangle \vdash_U \langle \Psi_1, \varphi\rangle \vdash_U \langle \Psi_2, \varphi\rangle \dots$$

Example 3. We want to test the formula $isin(r, L) \Rightarrow insert(r, L, L')$.

$$\Psi_0 = \{ \ (\{isin(r, L) \mathrel{|\!\!\sim} insert(r, L, L')\}, id) \ \}$$

$$\begin{aligned}\Psi_1 = \{ \ & (\emptyset, \sigma_1), \ (16)\\
& (\{eqr(x_0/s(y_0), e_0) \mathrel{|\!\!\sim} insert(x_0/s(y_0), e_0 :: l_0, l'_0),\\
& isin(x_0/s(y_0), l_0) \mathrel{|\!\!\sim} insert(x_0/s(y_0), e_0 :: l_0, l'_0)\}, \sigma_2), \ (17)\\
& (\{isin(x_0/s(y_0), e_0 :: l_0) \mathrel{|\!\!\sim} insert(x_0/s(y_0), l_0, l'_0)\}, \sigma_3), \ (19)\\
& (\emptyset, \sigma_4), \ (12)\\
& (\{isin(x_0/s(y_0), e_0 :: l_0) \mathrel{|\!\!\sim} eqr(x_0/s(y_0), e_0)\}, \sigma_5), \ (13)\\
& (\{isin(x_0/s(y_0), e_0 :: l_0) \mathrel{|\!\!\sim} ltr(x_0/s(y_0), e_0)\}, \sigma_6), \ (14)\\
& \{isin(x_0/s(y_0), e_0 :: l_0) \mathrel{|\!\!\sim} ltr(e_0, x_0/s(y_0)),\\
& isin(x_0/s(y_0), e_0 :: l_0) \mathrel{|\!\!\sim} insert(x_0/s(y_0), l_0, l'_0)\}, \sigma_7) \ (15) \ \}\end{aligned}$$

where

	r	L	L'	x	y	e	l	l'
σ_1	$x_0/s(y_0)$	$[\,]$		x_0	y_0			
σ_2	$x_0/s(y_0)$	$e_0 :: l_0$	l'_0	x_0	y_0	e_0	l_0	
σ_3	$x_0/s(y_0)$	l_0	l'_0	x_0	y_0		l_0	
σ_4	$x_0/s(y_0)$	$[\,]$	$x_0/s(y_0) :: [\,]$	x_0	y_0			
σ_5	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l_0$	x_0	y_0	e_0	l_0	
σ_6	$x_0/s(y_0)$	$e_0 :: l_0$	$x_0/s(y_0) :: e_0 :: l_0$	x_0	y_0	e_0	l_0	
σ_7	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l'_0$	x_0	y_0	e_0	l_0	l'_0

Each couple of Ψ_1 is labelled by the number of the axiom used for the unfolding of the initial formula.

The first couple $(\emptyset, \sigma_1)$ comes from the unification of the initial formula with the axiom $isin(x/s(y), [\,]) \mathrel{|\!\!\sim}$. Since $isin(r, L) \mathrel{|\!\!\sim} insert(r, L, L')$ with $r = x/s(y)$ and $L = [\,]$ is a direct consequence of this axiom, no constraint is generated but the substitution.

If L is not the empty list, the initial formula $isin(r, L) \mathrel{|\!\!\sim} insert(r, L, L')$ is true if and only if $L = L'$. Its unfolding when L is not empty will then lead to two kinds of constraints: those where $L = L'$ that will become test cases since they are consequences of the specification, and those where $L \neq L'$ that will not lead to test cases. For example, the fifth couple $(\{isin(x_0/s(y_0), e_0 :: l_0) \mathrel{|\!\!\sim} eqr(x_0/s(y_0), e_0)\}, \sigma_5)$ is a potential test case since $isin(x_0/s(y_0), e_0 :: l_0)$ and $eqr(x_0/s(y_0), e_0)$ are true simultaneously for any ground substitution. On the contrary, the sixth couple, whose constraint formula is $isin(x_0/s(y_0), e_0 ::$

$l_0) \mid\!\sim \mathit{ltr}(x_0/s(y_0), e_0)$, will never lead to a test case. Indeed, when $x_0/s(y_0)$ is in the list $e_0 :: l_0$, then it cannot be less than e_0, for any ground substitution.

The unfolding procedure cannot distinguish between these two kinds of constraints, however, before being submitted to the program, a ground substitution ρ is applied to constrained test purposes. Since by definition, $\rho(\psi)$ has to be a consequence of the specification, constraints where $L \neq L'$ will not be submitted as test cases to the program.

A second unfolding of, for example, the formula $\mathit{isin}(x_0/s(y_0), e_0 :: l_0) \mid\!\sim \mathit{eqr}(x_0/s(y_0), e_0)$ would lead to the following set:

$$\begin{array}{l}\{\ (\{\mathit{eqr}(x_0/s(y_0), e_0) \mid\!\sim \mathit{eqr}(x_0/s(y_0), e_0)\\ \quad \mathit{isin}(x_0/s(y_0), l_0) \mid\!\sim \mathit{eqr}(x_0/s(y_0), e_0)\}, \sigma'_1),\ (17)\\ \quad (\{\mathit{isin}(x_0/s(y_0), e_1 :: e_0 :: l_0) \mid\!\sim \mathit{eqr}(x_0/s(y_0), e_0)\}, \sigma'_2),\ (19)\\ \quad (\{\mathit{isin}(x_0/s(y_0), u_0/s(v_0) :: l_0) \mid\!\sim \mathit{mult}(x_0, s(v_0), n_0),\\ \quad \mathit{isin}(x_0/s(y_0), u_0/s(v_0) :: l_0) \mid\!\sim \mathit{mult}(u_0, s(y_0), n_0)\}, \sigma'_3),\ (10)\\ \quad (\{\mathit{isin}(x_0/s(y_0), l_0) \mid\!\sim\ \}, \sigma'_4)\ (17)\ \}\end{array}$$

The tautology $\mathit{eqr}(x_0/s(y_0), e_0) \mid\!\sim \mathit{eqr}(x_0/s(y_0), e_0)$ would be naturally deleted with the **Reduce** rule.

Here, our unfolding procedure has been defined in order to cover behaviours of one test purpose, represented by the formula φ. When we are interested in covering more widely the exhaustive set $Sp^\bullet \cap Obs$, a strategy consists in ordering quantifier-free first-order formula with respect to their length, as follows:

$$\Phi_0 = \{\ \mid\!\sim p(x_1, \ldots, x_n) \mid p : s_1 \times \ldots \times s_n \in P, \forall i, 1 \leq i \leq n, x_i \in V_{s_i}\}$$

$$\begin{array}{l}\Phi_{n+1} = \{p(x_1, \ldots, x_n), \Gamma \mid\!\sim \Delta, \quad \Gamma \mid\!\sim \Delta, p(x_1, \ldots, x_n) \mid \\ \qquad\qquad \Gamma \mid\!\sim \Delta \in \Phi_n, p : s_1 \times \ldots \times s_n \in P, \forall i, 1 \leq i \leq n, x_i \in V_{s_i}\}\end{array}$$

Then, to manage the size (often infinite) of $Sp^\bullet \cap Obs$, we start by choosing $k \in \mathbb{N}$, and then we apply for every i, $1 \leq i \leq k$, the above unfolding procedure to each $p(x_1, \ldots, x_n), \Gamma \mid\!\sim \Delta$ and $\Gamma \mid\!\sim \Delta, p(x_1, \ldots, x_n)$ belonging to Φ_i. Of course, this requires that signatures are finite so that each set Φ_i is finite too.

4.3 Soundness and Completeness

Here, we prove the two properties that make the unfolding procedure relevant for selection of appropriate test cases, i.e. that the selection criterion defined by the procedure is sound and complete for the initial test set we defined.

Test sets for quantifier-free formulae are naturally extended to sets of couples Ψ as follows:

$$T_{\Psi,\varphi} = \bigcup_{(\mathcal{C},\sigma) \in \Psi} T_{(\mathcal{C},\sigma),\varphi}$$

Theorem 2. *If* $\langle \Psi, \varphi \rangle \vdash_U \langle \Psi', \varphi \rangle$, *then* $T_{\Psi,\varphi} = T_{\Psi',\varphi}$.

The proof may be found in [15].

5 Conclusion

In this paper, we have extended a selection criterion, based on unfolding of positive conditional axioms in the algebraic specification setting, to quantifier-free first-order specifications. Our unfolding procedure consists in dividing an initial test set into subsets and then selecting test cases within each subset. We have then proved that this unfolding is complete. Moreover, we have shown that given a quantifier-free first-order specification Sp, $Sp^{\bullet} \cap Obs$ is an exhaustive set whatever the system under test is.

Research on this unfolding procedure is mainly continued on two aspects. First, we are specializing our unfolding procedure by handling equality (when it occurs) in a efficient way. Indeed equality often occurs in software specifications. When dealing with first-order logic, the axiomatization of equality leads to uniformly tackle this predicate as the others, without taking advantage of the efficient, natural and concise kind of reasoning which is attached to, namely, replacement of equal by equal. We are then adapting our unfolding procedure by defining it from sequent calculus $LK_=$ or $G_=$ [16]. Finally, our goal is to propose a framework of functional testing with selection criteria including primitive structuration, following [8, 13].

References

1. Le Gall, P., Arnould, A.: Formal specification and test: correctness and oracle. In: Haveraaen, M., Dahl, O.-J., Owe, O. (eds.) Recent Trends in Data Type Specification. LNCS, vol. 1130, pp. 342–358. Springer, Heidelberg (1996)
2. Gaudel, M.: Testing can be formal, too. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 82–96. Springer, Heidelberg (1995)
3. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software—Concepts and Tools 17(3), 103–120 (1996)
4. Bernot, G., Gaudel, M.C., Marre, B.: Software testing based on formal specifications: a theory and a tool. Software Engineering Journal 6(6), 387–405 (1991)
5. Marre, B.: Loft: a tool for assisting selection of test data sets from algebraic specifications. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 799–800. Springer, Heidelberg (1995)
6. Aiguier, M., Arnould, A., Boin, C., Le Gall, P., Marre, B.: Testing from algebraic specifications: Test data set selection by unfolding axioms. In: Grieskamp, W., Weise, C. (eds.) FATES 2005. LNCS, vol. 3997, pp. 203–217. Springer, Heidelberg (2006)
7. Machado, P.: On oracles for interpreting test results against algebraic specifications. In: Haeberer, A.M. (ed.) AMAST 1998. LNCS, vol. 1548, Springer, Heidelberg (1998)
8. Machado, P.: Testing from structured algebraic specifications. In: Rus, T. (ed.) AMAST 2000. LNCS, vol. 1816, pp. 529–544. Springer, Heidelberg (2000)
9. Arnould, A., Le Gall, P.: Test de conformité: une approche algébrique. Technique et Science Informatiques, Test de logiciel 21, 1219–1242 (2002)

10. Arnould, A., Le Gall, P., Marre, B.: Dynamic testing from bounded data type specifications. In: Hlawiczka, A., Simoncini, L., Silva, J.G.S. (eds.) Dependable Computing - EDCC-2. LNCS, vol. 1150, pp. 285–302. Springer, Heidelberg (1996)
11. Bernot, G.: Testing against formal specifications: a theoretical view. In: Abramsky, S. (ed.) TAPSOFT 1991, CCPSD 1991, and ADC-Talks 1991. LNCS, vol. 494, pp. 99–119. Springer, Heidelberg (1991)
12. Aiguier, M., Arnould, A., Le Gall, P.: Exhaustive test sets for algebraic specification correctness. Technical report, IBISC - Université d'Évry Val d'Essonne (2006)
13. Machado, P., Sannella, D.: Unit testing for CASL architectural specifications. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 506–518. Springer, Heidelberg (2002)
14. Hennicker, R., Wirsing, M., Bidoit, M.: Proof systems for structured specifications with observability operators. Theoretical Computer Science 173(2), 393–443 (1997)
15. Aiguier, M., Arnould, A., Le Gall, P., Longuet, D.: Test selection criteria for quantifier-free first-order specifications. Technical Report, -01, IBISC - Université d'Évry Val d'Essonne (2007), Available at `http://www.ibisc.fr/ dlonguet/Publications/RR-AALL07.pdf`
16. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Harper & Row, New York (1986)