Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Proof-Guided Test Selection from First-Order Specifications with Equality

Delphine Longuet · Marc Aiguier · Pascale Le Gall

the date of receipt and acceptance should be inserted later

Abstract This paper deals with test case selection from axiomatic specifications whose axioms are quantifier-free first-order formulas with equality. We first prove the existence of an ideal exhaustive test set to start the selection from. We then propose an extension of the test selection method called axiom unfolding, originally defined for algebraic specifications, to quantifier-free first-order specifications with equality. This method basically consists of a case analysis of the property under test (the test purpose) according to the specification axioms. It is based on a proof search for the different instances of the test purpose. Since the calculus is sound and complete, this allows us to provide a full coverage of this property. The generalisation we propose allows to deal with any kind of predicate (not only equality) and with any form of axiom and test purpose (not only equations or Horn clauses). Moreover, it improves our previous works with efficiently dealing with the equality predicate, thanks to the paramodulation rule.

Keywords Specification-based testing · quantifier-free first-order specifications · selection criteria · test purpose · axiom coverage · unfolding · proof tree normalisation

Introduction

Testing. Testing is a very common practice in the software development process. It is used at different steps of development to detect failures in a software system. The aim is not to prove the correctness of the system but only to ensure a certain confidence degree in its quality.

The principle of software testing is to execute the system on a finite subset of its possible inputs. The testing process is usually decomposed into three phases:

Delphine Longuet
Laboratoire Spécification et Vérification, ENS de Cachan, 61 avenue du Président Wilson, F-94235 Cachan Cedex
E-mail: longuet@lsv.ens-cachan.fr

Marc Aiguier · Pascale Le Gall
Laboratory of Mathematics Applied to Systems (MAS), École Centrale Paris, Grande Voie des Vignes, F-92295 Châtenay-Malabry
E-mail: marc.aiguier@ecp.fr, pascale.legall@ecp.fr

1. The *selection* of a relevant subset of the set of all possible inputs of the system, called a test set
2. The *submission* of this test set to the system
3. The *decision* of the success or the failure of the test set submission, also known as the "*oracle problem*".

Different strategies may be used to select test sets, thus defining several approaches to testing. The selection method called black-box testing is based on the use of a (formal or informal) specification as a reference object describing the intended behaviour of the system, without any knowledge of the implementation [7,11,8,18]. This allows in particular to use the same test set at different steps of the system development, if the specification does not change. Moreover, when the specification is formal, *i.e.* written in a mathematical formalism, both the selection phase and the oracle problem become easier to answer. On one hand, the specification of the different system behaviours (as axioms or paths for example) allows to select relevant test sets, by coverage of those behaviours. On the other hand, the oracle problem can be more easily solved since the result of a test can be deduced or even computed from the specification. The decision of the success of the test submission is then possible provided that there exists a means to compare the submission result and the expected output. Depending on the chosen formalism, if the system specification is executable for example, the selection and decision phases can even be automated.

Formal framework. The formalisation of the testing process allowed by the use of formal specifications requires some hypotheses and observability restrictions, concerning both the system under test and tests themselves. Test hypotheses on the system under test state that its behaviour can be described by a formal model. Moreover, it is assumed that test cases can be represented by formal objects that the system is able to understand. For instance in Tretmans' framework [31], since the specification is an Input Output Labelled Transition System (IOLTS), the system under test is assumed to be an IOLTS as well, in order to be able to compare the behaviours they both describe. Test cases are then traces, that are sequences of inputs and outputs, chosen from the specification and which the system must be able to perform.

Sometimes, observability restrictions are used to select test cases that can be interpreted as successful or not when performed by the system under test. For instance, in the framework of testing from equational specifications, where test cases are equations, the oracle problem comes from the distance between the abstract level of data type specification and concrete implementations. Intuitively, data types can be implemented according to concrete data representations such that it becomes difficult to state the appropriate abstraction/concretisation relation between the specified data types and implemented ones. The most common way to deal with this difficulty is to base the decision procedure of the success/failure only on the equality procedures provided in the programming language used to implement the system. Such sorts provided with a decidable predefined equality procedure are called observable. If we suppose that the implementation provides all the operations of the specification with an appropriate profile, then all ground terms used in the specification can be translated into computations within the implementation under test. These two assumptions explain that in the algebraic testing setting, test cases are usually represented as particular formulas, called observable formulas to refer to the oracle problem. These formulas are built from ground equations over observable sorts.

When such conditions (test hypotheses on the system and observability restrictions) are precisely stated, it becomes possible to formally define the testing activity [31, 19, 14, 17]. In particular, the notion of correctness of a system with respect to its specification can be defined up to these conditions, as well as important properties on test sets connecting the correctness to the success of a test set submission. A test set must at least not reject correct systems. Conversely, a test set should ideally reject any incorrect system. Unfortunately, it is well-known that in practice, testing is a very partial validation process: only few incorrect systems are rejected by a given test set. Nevertheless, if one would indefinitely continue to add new well-chosen test cases, one would expect that asymptotically any incorrect system could be rejected. In our framework, when testing from logical specifications, such an ideal test set is called exhaustive and is the starting point to the selection of a finite test set to submit to the system. In the approach called partition testing, the selection phase consists first in splitting an exhaustive test set into what we could call behavioural equivalence classes, and then in choosing one test case in each of these classes, thus building a finite test set which covers these behaviours. The first step is the *definition of selection criteria* characterising equivalence classes of behaviours, while the second step is the *generation of test cases* representing each of these behaviours. The assumption here, called the uniformity hypothesis, is that test cases in a class all are equivalent to make the system fail with respect to the behaviour they characterise.

Contribution. This paper addresses the definition of selection criteria in the framework of testing from first-order specifications with equality. Such specifications are classically used to specify standard software, i.e. software which computes results by manipulating complex data structures but with quite simple control structures.[1] Testing from algebraic specifications[2] has already been extensively studied [7, 19, 14, 6, 25, 5, 23, 24, 4, 1, 3]. Selection issues in particular have been investigated. The most popular selection method which has been studied is the one called *axiom unfolding* [6, 7, 25, 1]. It consists in dividing the exhaustive set into subsets according to criteria derived from the specification axioms. The fundamental idea behind this method is to use the well-known and efficient proof techniques of algebraic specifications for selecting test cases. Test cases must be consequences of the specification axioms, so they must be able to be proved as theorems in the theory defined by the specification. In other words, they can be deduced from the specification axioms using the calculus associated to the specification formalism. The idea behind axiom unfolding is to refine the initial exhaustive test set, replacing the initial property under test by the set of its different instances that can be deduced from the specification axioms. Intuitively, it consists of a case analysis. These particular cases of the initial property can themselves be unfolded and so on, leading to a finer and finer partition of the exhaustive test set. One must then prove that the obtained test set is as powerful as the initial one, that is no test is lost (the strategy is then said to be sound) and no test is added (the strategy is complete).

[1] In this sense, standard software differentiate from reactive systems, that are often characterised by the fact that they do not compute a result but rather maintain an interaction with their environment. Manipulated data structures are often very simple while control is very complex (parallel execution...). Many works have been dedicated to testing reactive and communicating systems, among which [20, 27, 31, 30, 12, 13].

[2] Algebraic specifications are a restriction of first-order specifications where the only predicate is equality and axioms are only equations or universal Horn clauses.

In this paper, we propose to extend the test selection method based on axiom unfolding to a larger class of axiomatic specifications: quantifier-free first-order specifications with equality. Compared to other works on axiom unfolding [25,1], the enlargement is twofold. First, we do not reduce atomic formulas to equations but also consider any kind of predicates. Secondly, formulas are not restricted to Horn clauses (called conditional positive formulas when dealing with equational logic).

This extension allows us to answer one of the main drawbacks of test selection methods for algebraic specifications: the very strong condition on systems needed to ensure the existence of an exhaustive test set. We actually proved in [3] that, when dealing with conditional positive specifications (i.e. equational specifications where axioms are under the form of a conjunction of equations implying an equation) and when test cases are equations only, an exhaustive test set only exists under a condition we called initiality, which requires the system under test to behave like the specification on the equations appearing in the premises of the axioms.[3] Since the system under test is supposed to be hidden in this framework of black-box testing, this condition can never be verified. We will show in this paper that when dealing with quantifier-free first-order specifications and ground first-order formulas as test cases, no condition on systems is needed to ensure the existence of an exhaustive test set.

Our first goal was to consider the whole classical first-order language. However, we showed in [3] that proving the existence of an exhaustive test set for such specifications boiled down to show the correctness of the system under test itself. Testing a formula of the form $\exists X, \varphi(X)$ would actually 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 astonishingly, exhibiting such a value would amount to simply prove the system with respect to the initial property. Thus, generally speaking, existential properties are not testable.

Related work. Other approaches of collaborations between proof and testing have recently been studied. For instance, testing can complement proof when a complete formal proof of the program correctness is not available [21]. Testing the weak parts of the proof, like pending lemmas, helps to gain confidence in the correctness of the program. In this approach, the testing process is not studied on its own, but rather as a complementary pragmatic approach to formal proof. Thus, exhaustivity or selection criteria are not investigated since the properties to test are given by the missing parts of the proof.

Closer to our own approach are the work [11,25,8,10,9]. The properties under test are directly derived from the axioms of the specification. A property under test is rewritten into a set of elementary properties, so that testing each of these properties separately is equivalent to testing the initial property. Intuitively, the rewriting of the initial property comes down to making a case analysis. This case analysis gives rise to a partition of the test set, such that at least a test case will be selected in each subset of the partition. For instance, either Boolean expressions are transformed into equivalent disjunctive normal forms, each conjunction becoming a set of the partition, or conditional axioms are reduced to equations by keeping only variable substitutions validating preconditions. In general, the family of elementary properties covers all the instances of the initial property, thus proving the soundness and the completeness of the selection criteria defined by the rewriting procedure for a given property. However,

[3] See [3] for a complete presentation of this result.

for all these works, axioms are necessarily the starting point of the testing process and more complicated properties involving several axioms are not considered. Since the aim of testing is to make the program under test fail, it is important to cover as many different behaviours as possible to have a hope to detect a failure. What we provide here is a more general approach allowing any formula to be tested, not only axioms and with non restricted form, given a quantifier-free first-order specification.

Some works on specification-based testing [23, 22] have already considered a similar class of formulas. 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 [19], we characterise 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.

Structure of the paper. The organisation of the paper follows the framework defined in [19], that we instantiate with the formalism of quantifier-free first-order logic. Section 1 recalls standard notations of quantifier-free first-order logic and gives the sequent calculus our selection method is based on. It also provides the example which is going to be used all along the paper. Section 2 sets the formal framework for testing from logical specifications: the underlying test hypotheses and observability restrictions are given, the notion of correctness of a system with respect to its specification is defined and an exhaustive test set for quantifier-free first-order specifications is characterised. In Section 3, the selection method by means of selection criteria is presented. We present our own selection method by axiom unfolding in Section 4 and prove its soundness and completeness.

This paper is an extension of [2] where we defined a test selection method based on axiom unfolding for first-order specifications but where equality was handled as any other predicate. We were thus loosing the natural, concise and efficient reasoning associated to equality: the replacement of equal by equal. We solve this problem here by using an additional inference rule called paramodulation. The paramodulation rule originally belongs to a calculus used as a refutational theorem proving method for first-order logic with equality [29]. Following the same principles that lead to resolution, Robinson and Wos introduced the paramodulation rule to replace several steps of resolution by an instantiation and the replacement of a subterm. In this paper, we then propose to define our test selection method based on axiom unfolding using a sequent calculus enriched by paramodulation.

1 Preliminaries

1.1 Quantifier-free first-order specifications with equality

Syntax. 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-indexed 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 formulas* are formulas of the form $t = t'$ with $t, t' \in T_\Sigma(V)_s$ or $p(t_1, \dots, t_n)$ 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 formulas and Boolean connectives $\neg$, $\wedge$, $\vee$ and $\Rightarrow$. As usual, variables of quantifier-free formulas 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-formulas.

A *specification* $Sp = (\Sigma, Ax)$ consists of a signature Σ and a set Ax of quantifier-free formulas built over Σ. Formulas in Ax are often called *axioms*.

Semantics. A Σ*-model* $\mathcal{M}$ is an S-indexed set $M = (M_s)_{s \in S}$ 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 set of all Σ-models.

Given a Σ-model $\mathcal{M}$, a Σ-assignment in M is any mapping $\nu : V \to M$ preserving sorts, i.e. $\nu(V_s) \subseteq M_s$. Assignments are naturally extended to terms with variables. A Σ-model $\mathcal{M}$ *satisfies* for an assignment ν a Σ-atomic formula $p(t_1, \dots, t_n)$ if and only if $(\nu(t_1), \dots, \nu(t_n)) \in p^{\mathcal{M}}$. The *satisfaction* of a Σ-formula φ for an assignment ν by $\mathcal{M}$, denoted by $\mathcal{M} \models_\nu \varphi$, is inductively defined on the structure of φ from the satisfaction for ν of atomic formulas of φ and using the classic semantic interpretation of Boolean connectives. $\mathcal{M}$ validates a formula φ, denoted by $\mathcal{M} \models \varphi$, if and only if for every assignment $\nu : V \to M$, $\mathcal{M} \models_\nu \varphi$.

Given $\Psi \subseteq For(\Sigma)$ and two Σ-models $\mathcal{M}$ and $\mathcal{M}'$, $\mathcal{M}$ is Ψ*-equivalent to* $\mathcal{M}'$, denoted by $\mathcal{M} \equiv_\Psi \mathcal{M}'$, if and only if we have: $\forall \varphi \in \Psi,\ \mathcal{M} \models \varphi \Longleftrightarrow \mathcal{M}' \models \varphi$.

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 subset of $Mod(\Sigma)$ whose elements are all Sp-models. A Σ-formula φ is a *semantic consequence* of a specification $Sp = (\Sigma, Ax)$, denoted by $Sp \models \varphi$, 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.

Herbrand model. Given a set of quantifier-free formulas $\Psi \subseteq For(\Sigma)$, 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 $T_\Sigma/_\approx$ where $\approx$ is the congruence on terms in T_Σ defined by $t \approx t'$ if and only if $\Psi \models t = t'$
  - whose operation meaning is defined for every operation $f : s_1 \times \dots \times s_n \to s \in F$ by the mapping $f^{\mathcal{H}_{T_\Sigma}} : ([t_1], \dots, [t_n]) \mapsto [f(t_1, \dots, t_n)]$
- such that $([t_1], \dots, [t_n]) \in p^{\mathcal{H}_{T_\Sigma}}$ if and only if $\Psi \models p(t_1, \dots, t_n)$.

This structure is valid since by definition, $\approx$ is a congruence (i.e. compatible with operations and predicates). It is easy to show that $\Psi \models \varphi \Leftrightarrow \mathcal{H}_{T_\Sigma} \models \varphi$ for every ground formula φ, and then $\mathcal{H}_{T_\Sigma} \in Mod((\Sigma, \Psi))$.

Calculus. A calculus for quantifier-free first-order specifications is defined by the following inference rules, where $\Gamma \vdash \Delta$ is a sequent such that Γ and Δ are two multisets of quantifier-free first-order formulas:

$$\frac{}{\Gamma, \varphi \vdash \Delta, \varphi}\,\mathsf{Taut} \qquad \frac{}{\Gamma \vdash \Delta}\,\mathsf{Ax} \quad (\Gamma \vdash \Delta \in Ax) \qquad \frac{}{\Gamma \vdash \Delta, t = t}\,\mathsf{Ref}$$

$$\frac{\Gamma \vdash \Delta, s = t \quad \Gamma' \vdash \Delta', \varphi[r]}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta'), \sigma(\varphi[t])}\,\mathsf{Para} \quad \begin{array}{c}(\sigma \text{ mgu of } s \text{ and } r,\\ r \text{ not a variable})\end{array}$$

$$\frac{\Gamma \vdash \Delta, \varphi}{\Gamma, \neg\varphi \vdash \Delta}\,\mathsf{Left}\text{-}\neg \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta, \neg\varphi}\,\mathsf{Right}\text{-}\neg$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}\,\mathsf{Left}\text{-}\wedge \qquad \frac{\Gamma \vdash \Delta, \varphi \quad \Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \wedge \psi}\,\mathsf{Right}\text{-}\wedge$$

$$\frac{\Gamma, \varphi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}\,\mathsf{Left}\text{-}\vee \qquad \frac{\Gamma \vdash \Delta, \varphi, \psi}{\Gamma \vdash \Delta, \varphi \vee \psi}\,\mathsf{Right}\text{-}\vee$$

$$\frac{\Gamma \vdash \Delta, \varphi \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \Rightarrow \psi \vdash \Delta}\,\mathsf{Left}\text{-}\Rightarrow \qquad \frac{\Gamma, \varphi \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \Rightarrow \psi}\,\mathsf{Right}\text{-}\Rightarrow$$

$$\frac{\Gamma \vdash \Delta}{\sigma(\Gamma) \vdash \sigma(\Delta)}\,\mathsf{Subs} \qquad \frac{\Gamma \vdash \Delta, \varphi \qquad \Gamma', \varphi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\mathsf{Cut}$$

Since we only consider quantifier-free formulas, rules for the introduction of quantifiers are useless.

Normalisation of sequents. Since the inference rules associated to Boolean connectives are reversible, they can be used from bottom to top to transform any sequent $\vdash \varphi$, where φ is a quantifier-free formula, into a set of sequents $\Gamma_i \vdash \Delta_i$ where every formula in Γ_i and Δ_i is atomic. Such sequents will be called *normalised sequents*.

This transformation is very useful, since we can show that every proof tree can be transformed into a proof tree of same conclusion and such that Para, Cut and Subs rules never occur under rule instances associated to Boolean connectives. This only holds under the assumption that axioms and tautologies introduced as leaves are normalised sequents. Then, all the sequents manipulated at the top of the proof tree are normalised, while the bottom of the proof tree only consists in applying rules to introduce Boolean connectives.

This transformation is obtained from basic transformations defined as rewriting rules where $\rightsquigarrow$ is the rewriting relation between elementary proof trees. For instance, the following rewriting rule allows to make any instance of the Cut rule go over an instance of the Left-$\neg$ rule occurring on its left-hand side:

$$\frac{\dfrac{\Gamma \vdash \Delta, \psi, \varphi}{\Gamma, \neg\varphi \vdash \Delta, \psi}\,\mathsf{Left}\text{-}\neg \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \neg\varphi \vdash \Delta, \Delta'}\,\mathsf{Cut} \rightsquigarrow \frac{\dfrac{\Gamma \vdash \Delta, \psi, \varphi \quad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta', \varphi}\,\mathsf{Cut}}{\Gamma, \Gamma', \neg\varphi \vdash \Delta, \Delta'}\,\mathsf{Left}\text{-}\neg$$

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

$$\mathsf{Ax}, \mathsf{Taut} > \mathsf{Para}, \mathsf{Cut}, \mathsf{Subs} > \mathsf{Left}\text{-}@, \mathsf{Right}\text{-}@, \text{ where } @ \in \{\neg, \wedge, \vee, \Rightarrow\}$$

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 set of normalised sequents, which allows to only deal with normalised sequents. Therefore, in the following, we will suppose that the specification axioms are given under the form of normalised sequents.

Remark. The normalised sequents can obviously be transformed into formulas in clausal form. Then the Cut rule can be replaced by the rule

$$\frac{\{\varphi\} \cup \neg\Gamma \cup \Delta \qquad \{\neg\varphi\} \cup \neg\Gamma' \cup \Delta'}{\neg\Gamma \cup \neg\Gamma' \cup \Delta \cup \Delta'}$$

We can easily show that in a proof tree, the substitution rule can always be applied just before the cut rule instead of just after, since it does not change the proof:

$$\dfrac{\dfrac{\Gamma \vdash \Delta, \varphi \quad \Gamma', \varphi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\mathsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Subs} \;\rightsquigarrow\; \dfrac{\dfrac{\Gamma \vdash \Delta, \varphi}{\sigma(\Gamma) \vdash \sigma(\Delta), \sigma(\varphi)}\,\mathsf{Subs} \quad \dfrac{\Gamma', \varphi \vdash \Delta'}{\sigma(\Gamma'), \sigma(\varphi) \vdash \sigma(\Delta')}\,\mathsf{Subs}}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}$$

It is then possible to combine the substitution and the cut rules to obtain the classical resolution rule:

$$\frac{\{\varphi\} \cup \neg\Gamma \cup \Delta \qquad \{\neg\varphi'\} \cup \neg\Gamma' \cup \Delta'}{\sigma(\neg\Gamma \cup \neg\Gamma' \cup \Delta \cup \Delta')}$$

where σ is a unifier of φ and φ'.

Actually, as we will see afterwards, this is the rule of resolution which is implemented in our unfolding algorithm. However, we believe that using the sequent calculus makes the completeness proof easier.

The soundness and completeness of the resolution calculus with paramodulation is well-known [29]. Therefore, the sequent calculus presented above is also sound and complete. Thus, for a given set of formulas, the set of syntactic and semantic consequences of these formulas coincide: theorems for this set of formulas are exactly its semantic consequences. From now on, we will then speak about theorems and semantic consequences without making any difference, in the framework of first-order logic.

1.2 Running example

By way of illustration, we give a specification of sorted lists of positive rationals, using the notations from the algebraic specification language Casl [26].

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

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, then the rational becomes the only element of the list; the first element of the list is equal to the rational to insert, then the element is not repeated; the first element of the list is greater than the rational to insert, 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 down to finding it at the head of the list or to searching it in the rest of the list.

spec $\textsc{RatList}$ =
  types $Nat ::= 0 \mid s(Nat)$
    $Rat ::= \_/\_(Nat, Nat)$
    $List ::= [\,] \mid \_ :: \_(Rat, List)$
  ops $add : Nat \times Nat \to Nat$
    $mult : Nat \times Nat \to Nat$
    $insert : Rat \times List \to List$
  preds $ltn : Nat \times Nat$
    $ltr : Rat \times Rat$
    $isin : Rat \times List$
  vars x, y, u, v: Nat; e: Rat; l: $List$
  • $add(x, 0) = x$
  • $add(x, s(y)) = s(add(x, y))$
  • $mult(x, 0) = 0$
  • $mult(x, s(y)) = add(x, mult(x, y))$
  • $ltn(0, s(x))$
  • $\neg ltn(x, 0)$
  • $ltn(s(x), s(y)) \Leftrightarrow ltn(x, y)$
  • $x/s(y) = u/s(v) \Leftrightarrow mult(x, s(v)) = mult(u, s(y))$
  • $ltr(x/s(y), u/s(v)) \Leftrightarrow ltn(mult(x, s(v)), mult(u, s(y)))$
  • $insert(x/s(y), [\,]) = x/s(y) :: [\,]$
  • $x/s(y) = e \Rightarrow insert(x/s(y), e :: l) = e :: l$
  • $ltr(x/s(y), e) \Rightarrow insert(x/s(y), e :: l) = x/s(y) :: e :: l$
  • $ltr(e, x/s(y)) \Rightarrow insert(x/s(y), e :: l) = e :: insert(x/s(y), l)$
  • $\neg isin(x/s(y), [\,])$
  • $isin(x/s(y), e :: l) \Leftrightarrow x/s(y) = e \vee isin(x/s(y), l)$
end

Axioms are then transformed into normalised sequents, as explained above. For example, the normalisation of the right-to-left implication of the axiom

$$isin(x/s(y), e :: l) \Leftrightarrow x/s(y) = e \vee isin(x/s(y), l)$$

leads to the two normalised sequents 19 and 20 (see below) thanks to the following proof tree:

$$\dfrac{\dfrac{x/s(y) = e \vdash isin(x/s(y), e :: l) \quad isin(x/s(y), l) \vdash isin(x/s(y), e :: l)}{x/s(y) = e \ \vee \ isin(x/s(y), l) \vdash isin(x/s(y), e :: l)}\,\textsf{Left-}\vee}{\vdash x/s(y) = e \ \vee \ isin(x/s(y), l) \ \Rightarrow \ isin(x/s(y), e :: l)}\,\textsf{Right-}\Rightarrow$$

Using the same transformation, the normalisation of the specification axioms leads to the following 20 sequents:

1. $\vdash add(x,0) = x$
2. $\vdash add(x,s(y)) = s(add(x,y))$
3. $\vdash mult(x,0) = 0$
4. $\vdash mult(x,s(y)) = add(x, mult(x,y))$
5. $\vdash ltn(0,s(x))$
6. $ltn(x,0) \vdash$
7. $ltn(s(x),s(y)) \vdash ltn(x,y)$
8. $ltn(x,y) \vdash ltn(s(x),s(y))$
9. $x/s(y) = u/s(v) \vdash mult(x,s(v)) = mult(u,s(y))$
10. $mult(x,s(v)) = mult(u,s(y)) \vdash x/s(y) = u/s(v)$
11. $ltr(x/s(y),u/s(v)) \vdash ltn(mult(x,s(v)), mult(u,s(y)))$
12. $ltn(mult(x,s(v)), mult(u,s(y))) \vdash ltr(x/s(y),u/s(v))$
13. $\vdash insert(x/s(y),[\,]) = x/s(y) :: [\,]$
14. $x/s(y) = e \vdash insert(x/s(y), e :: l) = e :: l$
15. $ltr(x/s(y),e) \vdash insert(x/s(y), e :: l) = x/s(y) :: e :: l$
16. $ltr(e,x/s(y)) \vdash insert(x/s(y), e :: l) = e :: insert(x/s(y), l)$
17. $isin(x/s(y),[\,]) \vdash$
18. $isin(x/s(y), e :: l) \vdash x/s(y) = e, isin(x/s(y), l)$
19. $x/s(y) = e \vdash isin(x/s(y), e :: l)$
20. $isin(x/s(y), l) \vdash isin(x/s(y), e :: l)$

From now on, the axioms of the specification RatList will only be used under this normalised form, we will only refer to these 20 normalised axioms.

2 Testing from logical specifications

2.1 Test hypotheses and observability restrictions

To be able to test a system against its specification, a general common semantic framework must be provided where the system and the specification behaviours can be compared. The system is assumed to implement sorts, operations and predicates given in the specification signature. In other words, given a logical specification (Σ, Ax), it gives an interpretation to symbols in Σ. Its behaviour is then considered as a Σ-model, that is an element of $Mod(\Sigma)$. For instance, an implementation of the RatList specification must implement naturals, rationals and lists, as well as the associated operations and predicates. Lists may be implemented with a list type in the programming language Caml or as arrays or linked lists in C for instance.

The system under test being a formal model, the properties it validates may be expressed as formulas. Test cases that are going to be submitted to the system are then formulas built over the specification signature, which are elements of $For(\Sigma)$. For instance, properties to test on an implementation of RatList could be:

$$insert(1/2, 1/4 :: 2/3 :: [\,]) = 1/4 :: 1/2 :: 2/3 :: [\,]$$
$$isin(2/6, 1/6 :: 1/3 :: 1/2 :: [\,])$$
$$isin(3/4, l) \Rightarrow isin(mult(3,n)/mult(4,n), l)$$
$$isin(x,l) \Rightarrow insert(x,l) = l$$

However, not every formula can be submitted to the system. The system must be able to evaluate the success or failure of the test case it is executed on. Actually, the third and fourth formulas above cannot be evaluated by a program, since they contain variables, that the program does not know how to interpret. Formulas that can be evaluated by the system under test are called observable formulas. Test cases then are chosen among observable formulas. When dealing with first-order specifications, the only requirement is that test cases must not contain non-instantiated variables. Test cases for the third and fourth properties above must then be ground instances of these formulas:

$$\begin{array}{l} isin(3/4, 1/2 :: 3/4 :: [\,]) \Rightarrow isin(mult(3,5)/mult(4,5), 1/2 :: 3/4 :: [\,]) \\ isin(3/4, 12/16 :: [\,]) \Rightarrow isin(mult(3,4)/mult(4,4), 12/16 :: [\,]) \\ isin(3/5, [\,]) \Rightarrow insert(3/5, [\,]) = [\,] \\ isin(2/6, 1/6 :: 1/3 :: [\,]) \Rightarrow insert(2/6, 1/6 :: 1/3 :: [\,]) = 1/6 :: 1/3 :: [\,] \end{array}$$

Observable formulas then are all ground formulas. The set of observable formulas is denoted by Obs.

We suppose in this paper that all sorts are observable, i.e. provided with a reliable decidable equality procedure in the system under test. Dealing with non observable sorts adds technical issues which would make the paper more difficult to read. Actually, two facts should be taken into account [15]: non observable equalities are observed through successive applications of functions leading to an observable result, usually called *observable contexts*; more embarrassing, one cannot in a pure black-box testing approach rely on some information based on non-observable equalities whose occurrences are in premisses of conditional equational formulas or more generally in negative positions in the axioms [22].

2.2 Correctness and exhaustivity

Under these hypotheses, it is possible to formally define the notion of correctness of a system with respect to its specification.

This notion is closely related to the interpretation of the submission of test cases. Since the system is considered as a formal model $\mathcal{S} \in Mod(\Sigma)$ and a test case is a ground formula $\varphi \in For(\Sigma)$, the success of the submission of φ to $\mathcal{S}$ is defined in terms of formula satisfaction: φ is *successful* for $\mathcal{S}$ if and only if $\mathcal{S} \models \varphi$. A test set T being a set of test cases, that is $T \subseteq For(\Sigma)$, T will be said successful for $\mathcal{S}$ if and only if every test case in T is successful: $\mathcal{S} \models T$ if and only if for all $\varphi \in T$, $\mathcal{S} \models \varphi$.

Following an observational approach [16], a system will be considered as a correct implementation of its specification if, as a model, it cannot be distinguished from a model of the specification. Since the system can only be observed through the observable formulas it satisfies, it is required to be equivalent to a model of the specification up to these observability restrictions.

Definition 1 (Correctness) A system $\mathcal{S}$ is *correct* for Sp via Obs, denoted by $Correct_{Obs}(\mathcal{S}, Sp)$, if and only if there exists a model $\mathcal{M}$ in $Mod(Sp)$ such that $\mathcal{M}$ validates exactly the same observable formulas as $\mathcal{S}$: $\mathcal{M} \equiv_{Obs} \mathcal{S}$.

This means that, to be correct, an implementation of the specification must at least satisfy all the observable formulas that the specification satisfies. The implementation cannot do nothing for instance. In other words, since by definition, all models of Sp satisfy the observable consequences of the specification, then a correct implementation should satisfy any such formula by construction. In the opposite case, if such a formula is not satisfied, the system cannot be considered as observationally equivalent to a model of the specification.

It is now possible to link the correctness of a system to the success of the submission of a test set. The first property required on a test set is that it does not reject correct systems. For instance, a correct implementation of RatList would be rejected by test cases like $ltr(1/2, 1/3)$ or $isin(2/5, [\,])$ which do not hold for the specification. It could also be rejected by a test case like $insert(1/2, 3/4 :: 1/3 :: [\,]) = 1/2 :: 3/4 :: 1/3 :: [\,]$ which is not part of the specification (the operation *insert* is not specified for unsorted lists). Therefore, such test cases are not wanted. A test set that does not reject correct systems is called *unbiased*. Thus, if a system fails for an unbiased test set, it is proved to be incorrect.

Conversely, if a test set rejects any incorrect system (but perhaps also correct ones), it is called *valid*. Then if a system passes a valid test set, it is proved to be correct. Incorrect implementations would not be rejected by tautologies like $1/2 = 1/2$ for instance. As another example, an incorrect implementation of the operation *isin* for which $isin(x, l)$ always holds would not be rejected by test cases like $isin(3/4, 3/4 :: [\,])$ or $isin(1/2, 1/3 :: 1/2 :: [\,])$, that do not cover all the specified behaviours. Such test sets are not sufficient to reject incorrect systems, they cannot form a valid test set.

An ideal test set would have at the same time the unbias and validity properties. The success of the submission of such a test set would actually prove the correctness of the system. Such a test set is called *exhaustive*.

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

$$\forall \mathcal{S} \in \mathcal{K}, \quad \mathcal{S} \models T \Leftrightarrow Correct_{Obs}(\mathcal{S}, Sp)$$

Let us point out that the notion of correctness is defined up to some class of models, denoted generically by $\mathcal{K}$. In the most general case, $\mathcal{K}$ is simply the whole class $Mod(\Sigma)$. It corresponds to the fact that no additionnal test hypotheses are made on the system, except the basic ones stating that the system can be modelled by a formal model. In practice, one can count on some particular knowledge on the system. For instance, if the specification includes the Boolean data type, it is quite natural to count on the fact that the two truth value are not equal in the system. All reliable information on the system which can be translated into formal hypotheses can be used to restrict the class of models to consider as a reference. The set $\mathcal{K}$ precisely captures in an abstract way all the hypotheses applied on the system.

The existence of an exhaustive test set ensures that it is possible to prove the correctness of the system under test with respect to its specification. To put it in a dual way, it ensures that for any incorrect system, there exists a test case making this system fail. Therefore, it is relevant to test this system with respect to its specification since its correctness can be asymptotically approached by submitting a potentially infinite test set. As a correctness reference, the exhaustive test set is then appropriate to start the selection of a finite test set of reasonable size.

2.3 Existence of an exhaustive test set

However, as we proved in [3], depending on the nature of the specification, on the observability restrictions and on the class of systems $\mathcal{K}$, an exhaustive test set does not necessarily exist.

The most natural ideal test set one could think of is the set of all ground instances of the specification axioms. However, it might happen that due to observability restrictions, some of these formulas cannot be chosen as test cases.

For instance, in practice, when dealing with algebraic specifications, test cases are restricted to ground equations on terms of particular sorts called observable sorts. These sorts are those equipped with an equality predicate in the programming language used to implement the system under test. In general, basic sorts like Booleans or integers are observable, but more complex data types like lists, trees or sets are not. Therefore, an axiom that would be an equation between terms of non-observable sort cannot be submitted as a test case since the oracle problem cannot be solved by the system under test. A classical answer to this problem is to assume that non-observable sorts are observable through observable contexts. For instance, a list L may be compared to another list L' by first comparing their first element $head(L)$ and $head(L')$, then their second one $head(tail(L))$ and $head(tail(L'))$, and so on. Then lists are most of the time observed through contexts $head(tail^n(\bullet))$ which is, in general, sufficient.

Another problem arises when specification axioms are not themselves equations, but for instance positive conditional formulas, that are a conjunction of equations implying an equation. In this case, axioms cannot be directly submitted as test cases, since they do not have the required form for test cases. A condition then has to be imposed to the system under test to ensure that the satisfaction of test cases as equations by the system is sufficient to prove the satisfaction of axioms. This condition is called initiality and imposes that the system under test behaves like the initial algebra (and so like the specification) on premises of axioms. The class $\mathcal{K}$ must then be restricted to systems satisfying this initiality condition. The interested reader may refer to [3] to a full study of conditions ensuring exhaustivity for algebraic specifications.

Moreover, as mentioned in the introduction, since the aim of testing is to make the system fail, the larger the exhaustive test set is, the better and the finer the detection of failures will be.

Among all possible test sets, the largest one is the set of semantic consequences of the specification. As a matter of fact, to be correct, the system under test must be observationally equivalent to a model of the specification, it must then satisfy exactly the same observable formulas as this model does. Now, formulas satisfied by all the models of a specification are by definition the semantic consequences of this specification, which set is denoted by $Sp^\bullet$. The system must then satisfy exactly the semantic consequences of the specification which are observable, i.e. formulas in $Sp^\bullet \cap Obs$.

We show here that considering a specification Sp with quantifier-free first-order axioms and the set of all ground first-order formulas as the set of observable formulas Obs, the exhaustivity of $Sp^\bullet \cap Obs$ holds without conditions on the system under test, that is $\mathcal{K} = Mod(\Sigma)$.

Theorem 1 *Let $Sp = (\Sigma, Ax)$ be a quantifier-free first-order specification and Obs be the set of all ground first-order formulas. Then $Sp^\bullet \cap Obs$ is exhaustive for $Mod(\Sigma)$.*

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

Define $Th(\mathcal{S}) = \{\varphi \in Obs \mid \mathcal{S} \models \varphi\}$. Let $\mathcal{H}_{T_\Sigma} \in Mod(\Sigma)$ be the Herbrand model of $Th(\mathcal{S})$. By definition, we have that $\mathcal{S} \equiv_{Obs} \mathcal{H}_{T_\Sigma}$. 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 assignment. By definition, $\nu(\varphi)$ is a ground formula. By hypothesis, $\mathcal{S} \models \nu(\varphi)$ and then $\mathcal{H}_{T_\Sigma} \models \nu(\varphi)$. We conclude that $\mathcal{H}_{T_\Sigma} \models_\nu \varphi$.

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

In [3], we generalised Theorem 1 by distinguishing observable and non-observable sorts. This requires a syntactical condition on the specification, where non-observable atomic formulas must only appear at "positive" positions in the axioms (φ is at a positive position and ψ at a negative position in $\varphi \wedge \neg\psi$ or in $\psi \Rightarrow \varphi$ for instance). See [3] for a complete presentation of this result. We do not consider non-observable sorts here for sake of simplicity.

3 Selection criteria

When it exists, the exhaustive test set is the starting point for the selection of a practical test set. In practice, experts apply selection criteria on a reference test set in order to extract a test set of reasonable size to submit to the system. The underlying idea is that all the test cases satisfying a certain selection criterion allow to detect the same class of incorrect systems. A selection criterion can then be seen as representing a fault model.

A selection criterion can be defined with respect to the size of the input data for instance, or with respect to the different specified behaviours of a given functionality. As an example, let us consider an implementation of the operation computing the multiplication of two square matrices of size $n \times n$, $n \geq 0$. One can consider that, if the implementation is not correct, it can be detected for small values of n, say less than 5. For $n \geq 5$, it can be assumed that the implementation will behave in a uniform way. This selection criteria is called the *regularity hypothesis* [7]. The implementation is assumed to behave in a uniform way for all input data of size greater than a given value. Therefore, in our example, it is sufficient to submit one test case for each value of n between 1 and 4, and one test case for $n \geq 5$, for instance with $n = 7$.

To give an example of another selection criterion, let us consider a function whose behaviour depends on a given threshold value t for one of its argument a. This function has three distinct behaviours, for $a < t$, $a = t$ and $a > t$. One can consider that the implementation of this function will behave in a uniform way for each of these sets of values. This selection criterion is called the *uniformity hypothesis* [7]. The implementation is assumed to behave in a uniform way for all input data respecting a given constraint. In other words, this hypothesis states that the test sets defined by these constraints form equivalent classes of behaviours, then submitting one test case for each of these classes is sufficient. Therefore, it is sufficient to submit one test case with $a < t$, one with $a = t$ and one with $a > t$ to test the function of our example.

These two hypotheses are very classical since [7]. They have been used in and adapted to other contexts, for instance in the framework of conformance testing for reactive systems [28]. Most of the existing selection methods are based on them. The selection method presented in this paper is in particular based on the uniformity hypothesis. It is obvious that this hypothesis makes more sense for small test sets than for very large ones. The aim then is to divide the initial exhaustive test set into smaller sets, so that this hypothesis can be applied in a meaningful way.

A classical method for selecting test sets with respect to a selection criterion C consists in dividing a reference test set T into a family of test subsets $\{T_i\}_{i \in I_{C(T)}}$ in such a way to preserve all test cases, *i.e.* $T = \bigcup_{i \in I_{C(T)}} T_i$. The application of a selection criterion associates a family of test sets to a given test set. All test cases in a test set T_i are supposed to be equivalent to detect systems which are incorrect with respect to the fault model captured by T_i. The application of a selection criterion to a given test set T allows to refine this test set. The obtained test subsets are more "specialised" than the initial test set, they correspond to more precise fault models than the one associated with T.

Definition 3 (Selection criterion) Let *Exh* be an exhaustive test set. A *selection criterion* C is a mapping[4] $\mathcal{P}(Exh) \to \mathcal{P}(\mathcal{P}(Exh))$.

For all $T \subseteq Exh$, $C(T)$ being a family of test sets $\{T_i\}_{i \in I_{C(T)}}$ where $I_{C(T)}$ is the set of indexes associated with the application of criterion C to T, we denote by $|C(T)|$ the set $\bigcup_{i \in I_{C(T)}} T_i$.

The construction of a test set relevant to a selection criterion must benefit from the division obtained by the application of this criterion. Test cases must be chosen so as not to loose any of the cases captured by the criterion.

Definition 4 (Satisfaction of a selection criterion) Let $T \subseteq Exh$ be a test set and C be a selection criterion. A test set T' *satisfies the criterion* C *applied to* T if and only if:

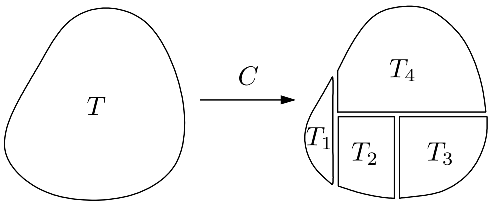
$$T' \subseteq |C(T)| \wedge \forall i \in I_{C(T)}, T_i \neq \emptyset \Rightarrow T' \cap T_i \neq \emptyset$$

A test set satisfying a selection criterion contains at least one test case of each subset T_i of the initial test set, when T_i is not empty. A selection criterion may then be considered as a coverage criterion, according to the way it divides the initial test set. It can be used to cover a particular aspect of the specification. In this paper, the definition of selection criteria will be based on the coverage of the specification axioms. This selection method is called *partition testing*.

Example 1 If we come back to the RatList specification, the *insert* operation is specified inductively by four axioms. Testing this operation comes down to testing the generic formula $insert(r, L) = L'$, where r, L, L' are variables. Now, testing a formula consists in finding input data, that are 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 the *insert* operation

$$T = \{\ insert(r, L) = L' \mid r, L, L' \in T_\Sigma,\ \ insert(r, L) = L' \in Sp^\bullet\ \}$$

can be split into four subsets:



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

$$\begin{array}{lll}
T_1 = \{\ insert(x/s(y), []) & = x/s(y) :: [] & \mid x, y \in T_\Sigma\ \} \\
T_2 = \{\ insert(x/s(y), e :: l) & = e :: l & \mid x/s(y) = e,\ x, y, e, l \in T_\Sigma\ \} \\
T_3 = \{\ insert(x/s(y), e :: l) & = x/s(y) :: e :: l & \mid ltr(x/s(y), e),\ x, y, e, l \in T_\Sigma\ \} \\
T_4 = \{\ insert(x/s(y), e :: l) & = e :: insert(x/s(y), l) & \mid ltr(e, x/s(y)),\ x, y, e, l \in T_\Sigma\ \}
\end{array}$$

The three last sets can be split again according to the axioms specifying the equality between rationals (in the case of test set T_2), the *ltr* predicate (T_3 and T_4) and the *insert* operation (T_4).

The relevance of a selection criterion is determined by the link between the initial test set and the family of test sets obtained by the application of this criterion.

Definition 5 (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 $|C(T)| \supseteq T$

These properties are essential for the definition of an appropriate selection criterion. The soundness of a criterion ensures that test cases are really selected among the initial test set. The application of the criterion does not add any new test case. Additional test cases may actually bias the test set, making a correct system fail. Reciprocally, if the selection criterion is complete, no test case of the initial test set is lost. If some test cases are missing, an incorrect system may pass the test set, while it should have failed on the missing test cases. A sound and complete selection criterion then has the property to preserve exactly all the test cases of the test set it divides, and then to preserve the unbias and the validity (and so the exhaustivity) of the initial test set.

4 Axiom unfolding

Building a test set to submit to the system consists in defining a method for dividing the initial test set into subsets and then, assuming the uniformity hypothesis, in choosing one test case in each of the obtained subsets. We will here present our method for defining relevant selection criteria in order to guide the final choice of the test cases. The application of the selection criteria will allow to refine the initial test set by characterising test subsets which respect given constraints on the input data.

4.1 Test sets for quantifier-free first-order formulas

The selection method we are going to present is called axiom unfolding and is based on a case analysis of the specification. This procedure was first defined for positive conditional specifications [6,7,25,1], i.e. axioms are formulas where a conjunction of equations implies another equation. In this setting, test cases are only equations. The exhaustive test set is the set of equations of the form $f(t_1, \ldots, t_n) = t$ where f is an operation of the signature, $t_1, \ldots, t_n, t$ are ground terms and such that this equation is a semantic consequence of the specification. To divide this initial test set comes down to dividing each test set associated to a given operation of the signature. The equation $f(t_1, \ldots, t_n) = t$ characterising this test set (when f is given) is called a test purpose. The operation f is specified by a certain number of axioms in the specification, which define the behaviour of this operation by cases, according to the different values of its inputs. The set of test cases for f will then be divided according to these different cases, that are the different axioms specifying this operation, as we showed in the previous section in Example 1.

Here, we generalise this procedure to quantifier-free first-order specifications. Since a test case may be any quantifier-free first-order formula, dividing the initial exhaustive test set comes down to dividing each test set associated to a given formula, chosen to be a test purpose. The test set associated to a formula naturally is the set of ground instances of this formula which are consequences of the specification.

Definition 6 (Test set for a formula) Let $Sp = (\Sigma, Ax)$ be a quantifier-free first-order specification. Let φ be a quantifier-free first-order formula, called *test purpose*. The *test set for* φ, denoted by T_φ, is the following set:

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

Note that the formula taken as a test purpose may be any formula, not necessarily a semantic consequence of the specification. However, only substitutions ρ such that $\rho(\varphi)$ is a semantic consequence of Sp will be built at the generation step.

Example 2 Here are some test purposes for the signature of the 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.*

$ltr(u, v)$. 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: $ltr(1/3, 1/2)$, $ltr(4/8, 4/6)$, *etc.*

$add(m, n) = mult(m, 2)$. Only cases where $m = n$ are semantic consequences of the specification, such as $add(2, 2) = mult(2, 2)$, $add(5, 5) = mult(5, 2)$, *etc.*

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

As we showed on Example 1, the test set for a formula is divided into subsets, that are themselves test sets for different instances of the initial test purpose, under certain constraints. Each of these test subsets can be characterised by a substitution σ, allowing to map the initial test purpose to its corresponding instance, and a set of constraints $\mathcal{C}$. For instance, the test set T_2 of Example 1

$$T_2 = \{\ insert(x/s(y), e :: l) = e :: l \mid x/s(y) = e,\ x, y, e, l \in T_\Sigma\ \}$$

may be characterised by the pair $(\mathcal{C}, \sigma)$ where

$$\mathcal{C} = \{x/s(y) = e\} \qquad \begin{aligned} \sigma : r &\mapsto x/s(y) \\ L &\mapsto e :: l \\ L' &\mapsto e :: l \end{aligned}$$

These test subsets, obtained by the application of a criterion, will be called constrained test sets.

Definition 7 (Constrained test set) Let $Sp = (\Sigma, Ax)$ be a quantifier-free first-order specification. Let φ be a quantifier-free first-order formula. Let $\mathcal{C}$ be a set of quantifier-free first-order formulas called *Σ-constraints* and $\sigma : V \to T_\Sigma(V)$ be a substitution. A *test set for φ constrained by $\mathcal{C}$ and σ*, denoted by $T_{(\mathcal{C},\sigma),\varphi}$, is the following set of ground formulas:

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

The pair $((\mathcal{C},\sigma),\varphi)$ is called a *constrained test purpose.*

Note that the test purpose of Definition 6 can be seen as the constrained test purpose $((\{\varphi\},\mathrm{Id}),\varphi)$.

Example 3 Examples of constrained test purposes may be the following:

$$\begin{array}{lll} \big(\,(\,\emptyset & ,\ \sigma : x \mapsto s(u) &),\ add(x,0) = x\,\big) \\ \big(\,(\,\{ltn(3,x)\} & ,\ \mathrm{Id} &),\ add(x,0) = x\,\big) \\ \big(\,(\,\{ltn(x,z)\} & ,\ \sigma : \begin{array}[t]{l} u \mapsto x/s(y) \\ v \mapsto z/s(y) \end{array} &),\ ltr(u,v)\,\big) \end{array}$$

As another example, to come back to Example 1 where we split the test set associated to $insert(r,L) = L'$ into four subsets, we can express each of them as constrained test purposes as follows:

$$\begin{array}{lll} \big(\,(\,\emptyset & ,\ \sigma_1 : \begin{array}[t]{l} r \mapsto x/s(y) \\ L \mapsto [\,] \\ L' \mapsto x/s(y) :: [\,] \end{array} &),\ insert(r,L) = L'\,\big) \\ \big(\,(\,\{x/s(y) = e\} & ,\ \sigma_2 : \begin{array}[t]{l} r \mapsto x/s(y) \\ L \mapsto e :: l \\ L' \mapsto e :: l \end{array} &),\ insert(r,L) = L'\,\big) \\ \big(\,(\,\{ltr(x/s(y),e)\} & ,\ \sigma_3 : \begin{array}[t]{l} r \mapsto x/s(y) \\ L \mapsto e :: l \\ L' \mapsto x/s(y) :: e :: l \end{array} &),\ insert(r,L) = L'\,\big) \\ \big(\,(\,\{ltr(e,x/s(y))\} & ,\ \sigma_4 : \begin{array}[t]{l} r \mapsto x/s(y) \\ L \mapsto e :: l \\ L' \mapsto e :: insert(x/s(y),l) \end{array} &),\ insert(r,L) = L'\,\big) \end{array}$$

Only this kind of constrained test purposes, 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 not constrained. The aim of the unfolding procedure is to replace it with a set of constrained test purposes, using the specification axioms. This procedure is an algorithm with the following inputs:

- a quantifier-free first-order specification $Sp = (\Sigma, Ax)$ where axioms of Ax have been transformed into normalised sequents

– a quantifier-free first-order formula φ seen as the initial test purpose.

As we already said, the initial test purpose φ can be seen as the constrained test purpose $((\{\varphi\}, \mathrm{Id}), \varphi)$, or even $((\mathcal{C}_0, \mathrm{Id}), \varphi)$ where $\mathcal{C}_0$ is the set of normalised sequents obtained from φ. Let Ψ_0 be the set containing the initial constraints of test purpose φ, the pair $(\mathcal{C}_0, \mathrm{Id})$. Constrained test sets for formulas are naturally extended to sets of pairs Ψ as follows:

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

The initial test set T_φ then is the set $T_{\Psi_0,\varphi}$.

The aim of the procedure is to divide this set according to the different cases in which formula φ holds. These cases correspond to the different instances of φ that can be proved as theorems, i.e. that can be deduced from the specification axioms using the calculus we gave in Section 1. So basically, the procedure searches for those proof trees that allow to deduce (instances of) the initial test purpose from the specification axioms. However, the aim is not to build the complete proofs of these instances of φ, but only to make a partition of $T_{\Psi_0,\varphi}$ increasingly fine. A first step in the construction of the proof tree of each instance will give us pending lemmas, constraints remaining to prove that, together with the right substitution, characterise each instance of φ. We will thus be able to replace Ψ_0, which contains only one constraint, with a set of constraints Ψ_1 characterising each instance of φ that can be proved from the axioms. The set Ψ_1 can itself be replaced with a bigger set Ψ_2 obtained from a second step in the construction of the previous proof trees, and so on. The procedure can be stopped at any moment, as soon as the tester is satisfied with the obtained partition.

Note that the procedure only intends to divide the test set associated to a given formula, by returning a set of constraints which characterise each set of the partition. The generation phase, not handled in this paper, consists in choosing one test case in each set of the partition (assuming the uniformity hypothesis) by solving the constraints associated to each set (which might be an issue in itself, due to the nature of these constraints).

The general case. As already explained, the procedure tries to divide the initial test set associated to a test purpose φ into test subsets by searching for proof trees of different instances of φ from the specification axioms. To achieve this purpose, it tries to unify the test purpose with an axiom, or more precisely, it tries to unify a subset of the test purpose's subformulas with a subset of an axiom's subformulas. Hence, if the test purpose is a normalised sequent of the form

$$\gamma_1, \ldots, \gamma_p, \ldots, \gamma_m \vdash \delta_1, \ldots, \delta_q, \ldots, \delta_n$$

the procedure tries to unify a subset of $\{\gamma_1, \ldots, \gamma_m, \delta_1, \ldots, \delta_n\}$ with a subset of the formulas of an axiom. Then it looks for a specification axiom of the form

$$\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k$$

such that it is possible to unify ψ_i and γ_i for all i, $1 \le i \le p$, and φ_i and δ_i for all i, $1 \le i \le q$.

Now we have to ensure that it is actually possible to prove (an instance of) the test purpose from this axiom. The inference rule which is fundamental at this stage is the Cut rule. It allows at the same time to delete the subformulas of the axiom that cannot be unified with subformulas of the test purpose, and to add the subformulas of the test purpose that do not exist in the axiom. To give a general picture, we have the following matching between the two formulas:

$$
\begin{array}{ll}
\text{Axiom} & \underbrace{\psi_1, \ldots, \psi_p}_{\text{unification}},\ \overbrace{\xi_1, \ldots, \xi_l}^{\text{to delete}}\ \vdash \underbrace{\varphi_1, \ldots, \varphi_q}_{\text{unification}},\ \overbrace{\zeta_1, \ldots, \zeta_k}^{\text{to delete}} \\
\text{Test purpose} & \overbrace{\gamma_1, \ldots, \gamma_p}, \underbrace{\gamma_{p+1}, \ldots, \gamma_m}_{\text{to add}} \vdash \overbrace{\delta_1, \ldots, \delta_q}, \underbrace{\delta_{q+1}, \ldots, \delta_n}_{\text{to add}}
\end{array}
$$

The additional formulas of the axiom and the missing formulas of the test purpose will be added or deleted thanks to applications of the cut rule. The proof will then consist of two steps.

1. First, we apply to the axiom the substitution σ that allows the unification.
2. Then, we cut one by one each formula of the axiom that cannot be unified with a subformula of the test purpose. This gives us a set of lemmas (the proof tree leaves) needed to complete the proof, the *constraints*. The last subformula to be cut also allows to add the missing subformulas. This is possible because in the premises of the rule, Γ, Δ and Γ', Δ' may be different.

The unifying substitution σ and the set of constraints given by the pending lemmas thus define a constrained test set characterising the instance of the test purpose we manage to prove.

If we denote by Γ the set of formulas $\{\gamma_1, \ldots, \gamma_p\}$, Γ' the set $\{\gamma_{p+1}, \ldots, \gamma_m\}$, Δ the set $\{\delta_1, \ldots, \delta_q\}$ and Δ' the set $\{\delta_{q+1}, \ldots, \delta_n\}$, we get a proof tree of the following form:

$$
\dfrac{\dfrac{\vdots}{\sigma(\Gamma') \vdash \sigma(\xi_l), \sigma(\Delta')} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \sigma(\xi_1)} \quad \dfrac{ST}{\sigma(\Gamma), \sigma(\xi_2), \ldots, \sigma(\xi_l) \vdash \sigma(\Delta)}\ \textsf{Cut}}{\vdots}\ \textsf{Cut}}{\sigma(\Gamma), \sigma(\xi_l) \vdash \sigma(\Delta)}\ \textsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta')}\ \textsf{Cut}
$$

where ST is the following subtree:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\textsf{Subs}\,\dfrac{\dfrac{}{\Gamma, \xi_1, \ldots, \xi_l \vdash \Delta, \zeta_1, \ldots, \zeta_k}\ \textsf{Ax}}{\sigma(\Gamma), \sigma(\xi_1), \ldots, \sigma(\xi_l) \vdash \sigma(\Delta), \sigma(\zeta_1), \ldots, \sigma(\zeta_k)} \quad \dfrac{\vdots}{\sigma(\zeta_1) \vdash}}{\sigma(\Gamma), \sigma(\xi_1), \ldots, \sigma(\xi_l) \vdash \sigma(\zeta_2), \ldots, \sigma(\zeta_k), \sigma(\Delta)}\ \textsf{Cut}}{\vdots}\ \textsf{Cut}}{\sigma(\Gamma), \sigma(\xi_1), \ldots, \sigma(\xi_l) \vdash \sigma(\Delta), \sigma(\zeta_k)}\ \textsf{Cut} \quad \dfrac{\vdots}{\sigma(\zeta_k) \vdash}}{\sigma(\Gamma), \sigma(\xi_1), \ldots, \sigma(\xi_l) \vdash \sigma(\Delta)}\ \textsf{Cut}
$$

As this proof tree shows, after having applied the substitution unifying some subformulas of the axiom with some subformulas of the test purpose (the formula to prove), $l + k$ applications of the Cut rule allow to delete the l subformulas of the left-hand side of the axiom and the k subformulas of its right-hand side, and moreover, allow to add the formulas of Γ' and Δ'.

The case of equality. In order to deal more efficiently with equality, this is the paramodulation rule that is used. When an axiom contains an equation, it can be used to replace a subterm of a formula with another subterm, equal to it according to this axiom. This rule makes the procedure more efficient to handle equality than standard first-order calculus, since it allows to use the replacement of equal by equal principle. So the procedure tries to unify a subterm of a subformula of the test purpose with one of the members of an equation in an axiom. Hence, if the test purpose is a normalised sequent of the form

$$\gamma_1,\ldots,\gamma_p,\ldots,\gamma_m \vdash \delta_1,\ldots,\delta_q,\ldots,\delta_n$$

the procedure tries to unify a subterm of δ_n (for instance) with one of the two sides of an equation appearing in an axiom. It also tries, as previously, to unify a subset of $\{\gamma_1,\ldots,\gamma_m,\delta_1,\ldots,\delta_{n-1}\}$ with a subset of the formulas of the axiom. Then it looks for a specification axiom of the form

$$\psi_1,\ldots,\psi_p,\xi_1,\ldots,\xi_l \vdash \varphi_1,\ldots,\varphi_q,\zeta_1,\ldots,\zeta_k, s = t$$

such that it is possible to unify a subterm of δ_n with t (for instance), ψ_i and γ_i for all i, $1 \leq i \leq p$, and φ_i and δ_i for all i, $1 \leq i \leq q$. The general picture is the following:

$$\begin{array}{llllll}
\text{Axiom} & \underbrace{\psi_1,\ldots,\psi_p}_{\text{unification}}, & \overbrace{\xi_1,\ldots,\xi_l}^{\text{to delete}} & \vdash \underbrace{\varphi_1,\ldots,\varphi_q}_{\text{unification}}, & \overbrace{\zeta_1,\ldots,\zeta_k}^{\text{to delete}}, & s = \underbrace{t}_{\text{unif.}} \\
\text{Test purpose} & \overbrace{\gamma_1,\ldots,\gamma_p}, & \underbrace{\gamma_{p+1},\ldots,\gamma_m}_{\text{to add}} & \vdash \overbrace{\delta_1,\ldots,\delta_q}, & \underbrace{\delta_{q+1},\ldots,\delta_{n-1}}_{\text{to add}}, & \delta_n \ \overbrace{[r]}
\end{array}$$

The test purpose can then be proved from this axiom in three steps:

1. We first apply to the axiom the substitution σ that allows the unification.
2. Then we apply the paramodulation rule twice to replace the subterm of δ_n with the other side of the equation s and to add the missing formulas.
3. Finally, we apply the Cut rule $k + l$ times as previously to delete the subformulas of the axiom that cannot be unified with subformulas of the test purpose.

We take the same notations as previously and additionnally, we denote by Ω the set of formulas $\{\xi_1,\ldots,\xi_l\}$ and by Λ the set $\{\zeta_1,\ldots,\zeta_k\}$. Since the subterm r of δ_n can be unified with t by σ, we get the following proof tree, which ends like the previous one with a serie of applications of the Cut rule:

$$\dfrac{\dfrac{\dfrac{\vdash \sigma(t)=\sigma(r) \qquad \dfrac{\mathsf{Subs}\dfrac{\dfrac{}{\Gamma,\Omega\vdash\Delta,\Lambda,s=t}\mathsf{Ax}}{\sigma(\Gamma),\sigma(\Omega)\vdash\sigma(\Delta),\sigma(\Lambda),\sigma(s)=\sigma(t)} \qquad \dfrac{\vdots}{\sigma(\Gamma')\vdash\sigma(\Delta'),\sigma(\delta_n[s])}}{\sigma(\Gamma),\sigma(\Omega),\sigma(\Gamma')\vdash\sigma(\Delta),\sigma(\Lambda),\sigma(\Delta'),\sigma(\delta_n[t])}\mathsf{Para}}{\sigma(\Gamma),\sigma(\Omega),\sigma(\Gamma')\vdash\sigma(\Delta),\sigma(\Lambda),\sigma(\Delta'),\sigma(\delta_n[r])}\mathsf{Para}}{\vdots}}{\sigma(\Gamma),\sigma(\Gamma')\vdash\sigma(\Delta),\sigma(\Delta'),\sigma(\delta_n[r])}\mathsf{Cut}$$

Let us note that the proof trees of the instances of the test purpose have, in both cases, a particular form. They actually respect the following structure:

– no instance of cut and paramodulation occurs over instances of substitution

– no instance of cut occurs in the left-hand side subtree of an instance of paramodulation
– there is no instance of paramodulation whose premises both are instances of paramodulation
– there is no instance of cut whose premises both are instances of cut.

This will be fundamental for the relevance of the method, as we will show in Theoreom 2.

The algorithm. The unfolding procedure is formally described by the following algorithm.[5] What it unfolds is a constraint ψ from a set of constraints $\mathcal{C}$ associated to some substitution σ in a pair of constraints $(\mathcal{C}, \sigma)$. The first set of constraints $\mathcal{C}_0$ only containing the initial test purpose, the procedure starts with unfolding this test purpose. It builds a set $Unf(\psi)$ representing the unfolding of ψ and containing all the pairs of constraints and substitution obtained by unfolding. Then it will unfold the constraints obtained from the unfolding of the test purpose, which will be considered themselves as test purposes, and so on. Given a constraint $\psi = \gamma_1, \dots, \gamma_m \vdash \delta_1, \dots, \delta_n$, the algorithm can be synthesised in the following way.

(Reduce) The first verification to make is whether some instances of the constraint are tautologies. If it is possible to unify some γ_i with some δ_j thanks to a substitution σ, then $\sigma(\psi)$ always holds and is useless. $\sigma(\psi)$ is then removed from the set of constraints associated to the corresponding instance of the test purpose.

(Unfold_Paramodulation) This is the part of the algorithm dealing with equality. If a subterm of a formula in the constraint can be unified with one of the members of an equation appearing in an axiom, and a subset of the other formulas of the constraint can be unified with a subset of the other formulas of the axiom, then it is possible to prove the constraint from this axiom with two applications of the paramodulation rule and a certain number of applications of the Cut rule. In this case, the set of constraints that is built is the set of all $\sigma'(\zeta_i) \vdash$ for i between 1 and k and all $\vdash \sigma'(\xi_i)$ for i between 1 and l, to which is added the lemma $\sigma'(\gamma_{p+1}), \dots, \sigma'(\gamma_m) \vdash \sigma'(\delta_{q+1}), \dots, \sigma'(\delta_{n-1}), \sigma'(\delta_n[s])$ needed to apply the paramodulation rule. It simply means that if a subterm of δ_n can be unified with t thanks to a substitution σ', then this subterm can be replaced with s in $\sigma'(\delta_n)$.

(Unfold_Cut) If the paramodulation rule cannot be used to unfold the constraint ψ, it is checked whether the Cut rule can be used alone. As explained before, if a part of the constraint can be unified with a part of an axiom, then we know that the constraint can be proved from this axiom with a certain number of applications of the Cut rule where, as in the previous case, each $\sigma'(\zeta_i) \vdash$ for i between 1 and k and each $\vdash \sigma'(\xi_i)$ for i between 1 and l is a lemma remaining to prove. One of those lemmas must bring the missing formulas, so $\sigma'(\xi_l)$ is in the context $\sigma'(\gamma_{p+1}), \dots, \sigma'(\gamma_m) \vdash \sigma'(\xi_l), \sigma'(\delta_{q+1}), \dots, \sigma'(\delta_n)$.

Then the procedure replaces the initial constraint ψ with each of the sets of constraints in $Unf(\psi)$. Each unification with an axiom leads to a pair (c, σ'), so the initial constraint ψ is replaced with as many sets of formulas as there are axioms with which it can be

[5] In the algorithm, $\delta_{n|_\omega}$ is the subterm appearing at position ω in δ_n, where positions in terms are defined by words over naturals, following the standard numbering of tree nodes.

Algorithm 1 Axiom unfolding

Inputs : quantifier-free first-order specification $Sp = (\Sigma, Ax)$, test purpose φ
Output : set of constraints Ψ

$\Psi \leftarrow \{(C_0, \mathrm{Id})\}$ where C_0 is the set of normalised sequents obtained from φ
loop
  Take (C, σ) from Ψ and remove it
  Take $\psi = \gamma_1, \ldots, \gamma_m \vdash \delta_1, \ldots, \delta_n$ from C and remove it
  $Unf(\psi) \leftarrow \emptyset$

(Reduce)
  if there exists $\sigma' \in T_\Sigma(V)^V$ mgu, $1 \leq i \leq m$ and $1 \leq j \leq n$
  such that $\sigma'(\gamma_i) = \sigma'(\delta_j)$ **then**
    Add $(\emptyset, \sigma')$ to $Unf(\psi)$

  else
    for all axioms $ax \in Ax$ **do**

    (Unfold_Paramodulation)
      if ax is of the form $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k, s = t$
      with $1 \leq p \leq m$, $1 \leq q \leq n$, and
      there exists $\sigma' \in T_\Sigma(V)^V$ mgu such that
      $\sigma'(\delta_{n_{|\omega}}) = \sigma'(t)$ and $\delta_{n_{|\omega}}$ is not a variable,
      for all $1 \leq i \leq p$, $\sigma'(\psi_i) = \sigma'(\gamma_i)$ and
      for all $1 \leq i \leq q$, $\sigma'(\varphi_i) = \sigma'(\delta_i)$ **then**
        $c \leftarrow \{\sigma'(\zeta_i) \vdash\ \}_{1 \leq i \leq k}$
          $\cup \{\ \vdash \sigma'(\xi_i)\}_{1 \leq i \leq l}$
          $\cup \{\sigma'(\gamma_{p+1}), \ldots, \sigma'(\gamma_m) \vdash \sigma'(\delta_{q+1}), \ldots, \sigma'(\delta_{n-1}), \sigma'(\delta_n[s])\}$
        Add (c, σ') to $Unf(\psi)$

    (Unfold_Cut)
      else if ax is of the form $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k$
      with $1 \leq p \leq m$, $1 \leq q \leq n$, and
      there exists $\sigma' \in T_\Sigma(V)^V$ mgu such that
      for all $1 \leq i \leq p$, $\sigma'(\psi_i) = \sigma'(\gamma_i)$ and
      for all $1 \leq i \leq q$, $\sigma'(\varphi_i) = \sigma'(\delta_i)$ **then**
        $c \leftarrow \{\sigma'(\zeta_i) \vdash\ \}_{1 \leq i \leq k}$
          $\cup \{\ \vdash \sigma'(\xi_i)\}_{1 \leq i \leq l-1}$
          $\cup \{\sigma'(\gamma_{p+1}), \ldots, \sigma'(\gamma_m) \vdash \sigma'(\xi_l), \sigma'(\delta_{q+1}), \ldots, \sigma'(\delta_n)\}$
        Add (c, σ') to $Unf(\psi)$

      end if
    end for
  end if

  Add $\displaystyle\bigcup_{(c,\sigma') \in Unf(\psi)} \{(\sigma'(\mathcal{C}) \cup c, \sigma' \circ \sigma)\}$ to Ψ
end loop

unified. The definition of $Unf(\psi)$ being based on unification, this set is computable if the specification has finitely many axioms.

Termination of the unfolding procedure is unlikely, since it is not checked, during its execution, whether the formula φ is a semantic consequence of the specification or not. Actually, this will be done during the generation phase, not handled in this paper. As we already explained, the aim of the unfolding procedure is not to find the complete proof of formula φ, but to make a partition of T_φ increasingly fine. Hence the procedure can be stopped at any moment, when the obtained partition is fine enough according to the judgement or the needs of the tester. The idea is to stretch further the execution of the procedure in order to make increasingly big proof trees whose remaining lemmas are constraints. If φ is not a semantic consequence of Sp, then this means that, among remaining lemmas, some of them do not hold, and then the associated test set is empty.

Given a formula ψ, the unfolding procedure defines the selection criterion C_ψ which maps $T_{(\mathcal{C},\sigma),\varphi}$ to the family of test sets $T_{(\sigma'(\mathcal{C}\setminus\{\psi\})\cup c,\sigma'\circ\sigma),\varphi}$ for each (c,σ') in $Unf(\psi)$ if ψ belongs to $\mathcal{C}$, and to itself otherwise. To ensure the relevance of this selection criterion, it must be shown that its application does not add new test cases to $T_{(\mathcal{C},\sigma),\varphi}$ (soundness) or remove test cases from it (completeness). These results are proved in the next subsection.

Coverage of the exhaustive test set. 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 formulas with respect to their length, as follows:

$$\Phi_0 = \left\{ \begin{array}{c} \vdash p(x_1,\ldots,x_n), \\ \vdash f(x_1,\ldots,x_n) = y \end{array} \;\middle|\; \begin{array}{l} p : s_1 \times \ldots \times s_n \in P, \\ f : s_1 \times \ldots \times s_n \to s \in F, \\ \forall i, 1 \leq i \leq n, x_i \in V_{s_i}, y \in V_s \end{array} \right\}$$

$$\Phi_{n+1} = \left\{ \begin{array}{c} p(x_1,\ldots,x_n), \Gamma \vdash \Delta, \\ f(x_1,\ldots,x_n) = y, \Gamma \vdash \Delta, \\ \Gamma \vdash \Delta, p(x_1,\ldots,x_n), \\ \Gamma \vdash \Delta, f(x_1,\ldots,x_n) = y \end{array} \;\middle|\; \begin{array}{l} \Gamma \vdash \Delta \in \Phi_n, \\ p : s_1 \times \ldots \times s_n \in P, \\ f : s_1 \times \ldots \times s_n \to s \in F, \\ \forall i, 1 \leq i \leq n, x_i \in V_{s_i}, y \in V_s \end{array} \right\}$$

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 formula belonging to Φ_i. Of course, this requires that signatures are finite so that each set Φ_i is finite too.

Example 4 First, to show the handling of equality with the paramodulation rule, we start with the formula of Example 1: $insert(r,L) = L'$. The associated constrained test purpose for this formula is

$$((\{\vdash insert(r,L) = L'\}, \mathrm{Id}), insert(r,L) = L')$$

We denote by Ψ_0 the set containing this first pair of constraints. After a loop of the algorithm, where the chosen constraint ψ is necessarily $\vdash insert(r,L) = L'$, we obtain the following set Ψ_1 (each pair is labelled by the number of the axiom used for the unfolding of the initial formula):

$$\begin{array}{lll}
\Psi_1 = \{\ (\{\ \vdash x_0/s(y_0) :: [\,] = L'\} & , \sigma_1 : r \mapsto x_0/s(y_0)\ ) & (13)\\
& \qquad L \mapsto [\,] & \\[1ex]
\qquad (\{\ \vdash x_0/s(y_0) = e_0, & , \sigma_2 : r \mapsto x_0/s(y_0)\ ) & (14)\\
\qquad\quad \vdash e_0 :: l_0 = L'\} & \qquad L \mapsto e_0 :: l_0 & \\[1ex]
\qquad (\{\ \vdash ltr(x_0/s(y_0), e_0), & , \sigma_3 : r \mapsto x_0/s(y_0)\ ) & (15)\\
\qquad\quad \vdash x_0/s(y_0) :: e_0 :: l_0 = L'\} & \qquad L \mapsto e_0 :: l_0 & \\[1ex]
\qquad (\{\ \vdash ltr(e_0, x_0/s(y_0)), & , \sigma_4 : r \mapsto x_0/s(y_0)\ )\} & (16)\\
\qquad\quad \vdash e_0 :: insert(x_0/s(y_0), l_0) = L'\} & \qquad L \mapsto e_0 :: l_0 &
\end{array}$$

We get the same four sets as in Examples 1 and 3, although they are expressed a bit differently. The four sets of Example 3 are what we would have obtained with the application of the **(Unfold_Cut)** part of the algorithm. We can see here that using paramodulation results in simpler substitutions and more constraints, which allows to unfold more finely the initial formula. For instance, the last constraint in the fourth set $\vdash e_0 :: insert(x_0/s(y_0), l_0) = L'$ was kept implicit in the substitution given in the fourth set of Example 3, which prevents from unfolding it as we can do here.

Actually, we now show the unfolding of this last constraint. We get the following set, which replaces the fourth set above (the new substitution is already composed with the previous one):

$$\begin{array}{lll}
\{\ (\{\ \vdash ltr(e_0, x_0/s(y_0)), & , \sigma'_1 : r \mapsto x_0/s(y_0)\quad) & (13)\\
\qquad \vdash e_0 :: x_0/s(y_0) :: [\,] = L'\} & \qquad L \mapsto e_0 :: [\,] & \\[1ex]
\quad (\{\ \vdash ltr(e_0, x_0/s(y_0)), & , \sigma'_2 : r \mapsto x_0/s(y_0)\quad) & (14)\\
\qquad \vdash x_0/s(y_0) = e_1, & \qquad L \mapsto e_0 :: e_1 :: l_1 & \\
\qquad \vdash e_0 :: e_1 :: l_1 = L'\} & & \\[1ex]
\quad (\{\ \vdash ltr(e_0, x_0/s(y_0)), & , \sigma'_3 : r \mapsto x_0/s(y_0)\quad) & (15)\\
\qquad \vdash ltr(x_0/s(y_0), e_1), & \qquad L \mapsto e_0 :: e_1 :: l_1 & \\
\qquad \vdash e_0 :: x_0/s(y_0) :: e_1 :: l_1 = L'\} & & \\[1ex]
\quad (\{\ \vdash ltr(e_0, x_0/s(y_0)), & , \sigma'_4 : r \mapsto x_0/s(y_0)\quad)\} & (16)\\
\qquad \vdash ltr(e_1, x_0/s(y_0)), & \qquad L \mapsto e_0 :: e_1 :: l_1 & \\
\qquad \vdash e_0 :: e_1 :: insert(x_0/s(y_0), l_1) = L'\} & &
\end{array}$$

To see that these sets correspond to what we expect from the procedure, we can write them in a more natural way:

$$\begin{array}{ll}
\{insert(x_0/s(y_0), e_0 :: [\,]) = e_0 :: x_0/s(y_0) :: [\,] & \mid ltr(e_0, x_0/s(y_0))\}\\
\{insert(x_0/s(y_0), e_0 :: e_1 :: l_1) = e_0 :: e_1 :: l_1 & \mid ltr(e_0, x_0/s(y_0))\\
& \ \ x_0/s(y_0) = e_1\}\\
\{insert(x_0/s(y_0), e_0 :: e_1 :: l_1) = e_0 :: x_0/s(y_0) :: e_1 :: l_1 & \mid ltr(e_0, x_0/s(y_0))\\
& \ \ ltr(x_0/s(y_0), e_1)\}\\
\{insert(x_0/s(y_0), e_0 :: e_1 :: l_1) = e_0 :: e_1 :: insert(x_0/s(y_0), l_1) & \mid ltr(e_0, x_0/s(y_0))\\
& \ \ ltr(e_1, x_0/s(y_0))\}
\end{array}$$

We could have unfolded any other constraint of one of the first four sets, according to the axioms specifying the predicates over rationals for instance.

Example 5 To show a complete example where cut and paramodulation are used, we take the formula $isin(r, L) \Rightarrow insert(r, L) = L'$. This formula does not hold for every substitution of the variables. If L is not the empty list, the instances where $L = L'$ only are consequences of the specification (as well as instances where $isin(r, L)$ does not hold, but these one are of no interest for our purpose). The unfolding of this formula will then lead to two kinds of constraints: those where $L = L'$ that will actually become test cases since they are consequences of the specification, and those where $L \neq L'$ that will not lead to test cases. The procedure cannot distinguish between these two kinds of constraints. However, before being submitted to the system, a ground substitution ρ is applied to constrained test purposes. Since by definition, $\rho(\varphi)$ has to be a consequence of the specification, the constraints where $L \neq L'$ will not be submitted as test cases to the system.

The associated constrained test purpose for this formula is

$$((\{isin(r, L) \vdash insert(r, L) = L'\}, \mathrm{Id}), isin(r, L) \Rightarrow insert(r, L) = L')$$

We denote by Ψ_0 the set containing this first pair of constraints. The set Ψ_1 we get after one loop of the algorithm is the following:

$$\{(\{\ isin(x_0/s(y_0), [\,]) \vdash x_0/s(y_0) :: [\,] = L'\} \qquad , \sigma_1 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto [\,] \end{array}) \tag{13}$$

$$(\{\ \vdash x_0/s(y_0) = e_0,\ isin(x_0/s(y_0), e_0 :: l_0) \vdash e_0 :: l_0 = L'\} \qquad , \sigma_2 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto e_0 :: l_0 \end{array}) \tag{14}$$

$$(\{\ \vdash ltr(x_0/s(y_0), e_0),\ isin(x_0/s(y_0), e_0 :: l_0) \vdash x_0/s(y_0) :: e_0 :: l_0 = L'\} \qquad , \sigma_3 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto e_0 :: l_0 \end{array}) \tag{15}$$

$$(\{\ \vdash ltr(e_0, x_0/s(y_0)),\ isin(x_0/s(y_0), e_0 :: l_0) \vdash e_0 :: insert(x_0/s(y_0), l_0) = L'\} \qquad , \sigma_4 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto e_0 :: l_0 \end{array}) \tag{16}$$

$$(\{\ \emptyset \qquad , \sigma_5 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto [\,] \end{array}) \tag{17}$$

$$(\{\ x_0/s(y_0) = e_0 \vdash,\ isin(x_0/s(y_0), l_0) \vdash insert(x_0/s(y_0), e_0 :: l_0) = L'\} \qquad , \sigma_6 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto e_0 :: l_0 \end{array}) \tag{18}$$

$$(\{\ isin(x_0/s(y_0), e :: l_0) \vdash insert(x_0/s(y_0), l_0) = L'\} \qquad , \sigma_7 : \begin{array}{l} r \mapsto x_0/s(y_0) \\ L \mapsto l_0 \end{array})\} \tag{20}$$

The first four sets are almost the same as in the previous example, up to the left-hand side $isin(x_0/s(y_0), e_0 :: l_0)$, since it is the **(Unfold_Paramodulation)** part of the algorithm that is used for the unfolding. The last three sets are built using the **(Unfold_Cut)** part on the axioms specifying the membership predicate *isin*.

Intuitively, the part of the algorithm using paramodulation will be used to unfold formulas involving operations, while the part using the Cut rule will be used to unfold formulas with predicates.

4.3 Properties of the selection criterion

Here, we prove the two properties that make the unfolding procedure relevant for the 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.

To prove the soundness of the procedure comes down to proving that the instance $\sigma'(\varphi)$ of the initial formula φ can be derived from the set of constraints c and the axiom with which it has been unified. Thus we prove that the test set obtained by the application of the procedure does not contain new test cases, since it is only composed of instances of the initial test purpose.

To prove the completeness, we prove that all the possible instances of the test purpose can be proved with a proof tree of the form we showed earlier, and that the procedure generates all possible constraints for proving this instance. We thus prove that no test case is lost. Actually, we can observe that our unfolding procedure defines a proof search strategy that enables to limit the search space to the class of proof trees having the following structure:

- no instance of cut and paramodulation occurs over instances of substitution
- no instance of cut occurs in the left-hand side subtree of an instance of paramodulation
- there is no instance of paramodulation whose premises both are instances of paramodulation
- there is no instance of cut whose premises both are instances of cut.

We then have to prove that the derivability defined by our unfolding strategy coincides with the full derivability. We then define basic transformations to rewrite proof trees into ones having the above structure, and show that the induced global proof tree transformation is weakly normalising.

Theorem 2 (Soundness and completeness) *Let φ be a quantifier-free first-order formula, $\mathcal{C}$ a set of constraints and $\sigma : V \to T_\Sigma(V)$ a substitution. Let $\psi \in \mathcal{C}$. The* selection criterion for ψ is sound and complete *for the test set for φ constrained by $\mathcal{C}$ and σ:*

$$|C_\psi(T_{(\mathcal{C},\sigma),\varphi})| = T_{(\mathcal{C},\sigma),\varphi}$$

Proof **(Soundness)** Let us prove that $|C_\psi(T_{(\mathcal{C},\sigma),\varphi})| \subseteq T_{(\mathcal{C},\sigma),\varphi}$.

By definition, it is sufficient to prove that for each $(\mathcal{C},\sigma) \in \Psi$, for each $\psi \in \mathcal{C}$, for each $(c,\sigma') \in \mathit{Unf}(\psi)$, $T_{(c,\sigma'\circ\sigma),\varphi} \subseteq T_{(\{\psi\},\sigma),\varphi}$. We then have to prove that for each ground substitution $\rho : V \to T_\Sigma$ such that $\mathit{Sp} \models \rho(\chi)$, for each $\chi \in c$, there exists $\rho' : V \to T_\Sigma$ such that $\mathit{Sp} \models \rho'(\psi)$. A first case to consider is the **Reduce** case. Then two cases have to be considered, depending on the form of the axiom the constraint ψ has been unified with.

Case 1 - Reduce. If there exists a substitution σ' such that $\sigma'(\psi)$ is a tautology, then any ground instance $\rho\sigma'(\psi)$ can be proved with no additional constraints.

Case 2 - Unfold_Cut. Let us assume that the formula ψ is of the form $\gamma_1, \ldots, \gamma_m \vdash \delta_1, \ldots, \delta_n$, and that the set c such that $(c,\sigma') \in \mathit{Unf}(\psi)$ is of the form

$$\begin{array}{l}\{\sigma'(\zeta_i) \vdash \}_{1 \le i \le k} \\ \cup \{ \vdash \sigma'(\xi_i)\}_{1 \le i \le l-1} \\ \cup \{\sigma'(\gamma_{p+1}), \ldots, \sigma'(\gamma_m) \vdash \sigma'(\xi_l), \sigma'(\delta_{q+1}), \ldots, \sigma'(\delta_n)\}\end{array}$$

where $1 \leq p \leq m$ and $1 \leq q \leq n$ are such that $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k \in Ax$, $\sigma'(\psi_i) = \sigma'(\gamma_i)$ for each $1 \leq i \leq p$ and $\sigma'(\varphi_i) = \sigma'(\delta_i)$ for each $1 \leq i \leq q$. We then have the following proof tree, where $\Gamma = \{\psi_1, \ldots, \psi_p\}$, $\Delta = \{\varphi_1, \ldots, \varphi_q\}$, $\Gamma' = \{\gamma_{p+1}, \ldots, \gamma_m\}$, $\Delta' = \{\delta_{q+1}, \ldots, \delta_n\}$, for each i, $1 \leq i \leq l$, $\Omega_i = \{\xi_i, \ldots, \xi_l\}$ and for each i, $1 \leq i \leq k$, $\Lambda_i = \{\zeta_i, \ldots, \zeta_k\}$. The composition $\sigma \circ \sigma'$ of two substitutions $\sigma' : V \to T_\Sigma(V)$ and $\sigma : T_\Sigma(V) \to T_\Sigma(V)$, applied to a formula φ, is denoted by $\sigma\sigma'(\varphi)$.

$$\dfrac{\dfrac{\vdots}{\rho\sigma'(\Gamma') \vdash \rho\sigma'(\xi_l), \rho\sigma'(\Delta')} \quad \dfrac{\begin{matrix}\dfrac{\dfrac{\vdots}{\vdash \rho\sigma'(\xi_2)} \quad \dfrac{\dfrac{\vdots}{\vdash \rho\sigma'(\xi_1)} \quad ST}{\rho\sigma'(\Gamma), \rho\sigma'(\Omega_2) \vdash \rho\sigma'(\Delta)}\,\mathsf{Cut}}{}\,\mathsf{Cut} \\ \vdots\end{matrix}}{\rho\sigma'(\Gamma), \rho\sigma'(\Omega_l) \vdash \rho\sigma'(\Delta)}\,\mathsf{Cut}}{\rho\sigma'(\Gamma), \rho\sigma'(\Gamma') \vdash \rho\sigma'(\Delta), \rho\sigma'(\Delta')}\,\mathsf{Cut}$$

where ST is the following subtree:

$$\dfrac{\begin{matrix}\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \Omega_1 \vdash \Lambda_1, \Delta}\,\mathsf{Ax}}{\rho\sigma'(\Gamma), \rho\sigma'(\Omega_1) \vdash \rho\sigma'(\Lambda_1), \rho\sigma'(\Delta)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\rho\sigma'(\zeta_1) \vdash}}{\rho\sigma'(\Gamma), \rho\sigma'(\Omega_1) \vdash \rho\sigma'(\Lambda_2), \rho\sigma'(\Delta)}\,\mathsf{Cut} \quad \dfrac{\vdots}{\rho\sigma'(\zeta_2) \vdash}}{}\,\mathsf{Cut} \\ \vdots\end{matrix}}{\rho\sigma'(\Gamma), \rho\sigma'(\Omega_1) \vdash \rho\sigma'(\Delta)}\,\mathsf{Cut}$$

Case 3 - Unfold_Paramodulation. Let us assume that the formula ψ is of the form $\gamma_1, \ldots, \gamma_m \vdash \delta_1, \ldots, \delta_n$, and that the set c such that $(c, \sigma') \in \mathit{Unf}(\psi)$ is of the form

$$\begin{array}{l} \{\sigma'(\zeta_i) \vdash \}_{1 \leq i \leq k} \\ \cup\{ \vdash \sigma'(\xi_i)\}_{1 \leq i \leq l} \\ \cup\{\sigma'(\gamma_{p+1}), \ldots, \sigma'(\gamma_m) \vdash \sigma'(\delta_{q+1}), \ldots, \sigma'(\delta_{n-1}), \sigma'(\delta_n[s])\} \end{array}$$

where $1 \leq p \leq m$ and $1 \leq q \leq n$ are such that $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k, s = t \in Ax$, $\sigma'(\delta_{n_{|\omega}}) = \sigma'(t)$, $\sigma'(\psi_i) = \sigma'(\gamma_i)$ for each $1 \leq i \leq p$ and $\sigma'(\varphi_i) = \sigma'(\delta_i)$ for each $1 \leq i \leq q$. We then have the following proof tree, where we follow the same notations as previously except that Δ' is the set $\{\delta_{q+1}, \ldots, \delta_{n-1}\}$:

$$\dfrac{\dfrac{\vdots}{\vdash \rho\sigma'(\xi_l)} \quad \dfrac{\begin{matrix}\dfrac{\dfrac{\vdots}{\vdash \rho\sigma'(\xi_2)} \quad \dfrac{\dfrac{\vdots}{\vdash \rho\sigma'(\xi_1)} \quad ST}{\rho\sigma'(\Gamma), \rho\sigma'(\Gamma'), \rho\sigma'(\Omega_2) \vdash \rho\sigma'(\Delta), \rho\sigma'(\Delta'), \rho\sigma'(\delta_n[r])}\,\mathsf{Cut}}{}\,\mathsf{Cut} \\ \vdots\end{matrix}}{\rho\sigma'(\Gamma), \rho\sigma'(\Gamma'), \rho\sigma'(\Omega_l) \vdash \rho\sigma'(\Delta), \rho\sigma'(\Delta'), \rho\sigma'(\delta_n[r])}\,\mathsf{Cut}}{\rho\sigma'(\Gamma), \rho\sigma'(\Gamma') \vdash \rho\sigma'(\Delta), \rho\sigma'(\Delta'), \rho\sigma'(\delta_n[r])}\,\mathsf{Cut}$$

where ST is the following subtree:

$$\dfrac{\begin{matrix}\dfrac{\dfrac{ST' \quad \dfrac{\vdots}{\rho\sigma'(\zeta_1) \vdash}}{\rho\sigma'(\Gamma), \rho\sigma'(\Gamma'), \rho\sigma'(\Omega_1) \vdash \rho\sigma'(\Lambda_2), \rho\sigma'(\Delta), \rho\sigma'(\Delta'), \rho\sigma'(\delta_n[r])}\,\mathsf{Cut} \quad \dfrac{\vdots}{\rho\sigma'(\zeta_2) \vdash}}{}\,\mathsf{Cut} \\ \vdots\end{matrix}}{\rho\sigma'(\Gamma), \rho\sigma'(\Gamma'), \rho\sigma'(\Omega_1) \vdash \rho\sigma'(\Delta), \rho\sigma'(\Delta'), \rho\sigma'(\delta_n[r])}\,\mathsf{Cut}$$

and *ST'* is the following subtree:

$$\dfrac{\rho\sigma'(t)=\rho\sigma'(r) \qquad \dfrac{\mathsf{Subs}\,\dfrac{\dfrac{}{\Gamma,\Omega_1\vdash\Lambda_1,\Delta,s=t}\,\mathsf{Ax}}{\rho\sigma'(\Gamma),\rho\sigma'(\Omega_1)\vdash\rho\sigma'(\Lambda_1),\rho\sigma'(\Delta),\rho\sigma'(s)=\rho\sigma'(t)} \qquad \dfrac{\vdots}{\rho\sigma'(\Gamma')\vdash\rho\sigma'(\Delta'),\rho\sigma'(\delta_n[s])}}{\rho\sigma'(\Gamma),\rho\sigma'(\Gamma'),\rho\sigma'(\Omega_1)\vdash\rho\sigma'(\Lambda_1),\rho\sigma'(\Delta),\rho\sigma'(\Delta'),\rho\sigma'(\delta_n[t])}\,\mathsf{Para}}{\rho\sigma'(\Gamma),\rho\sigma'(\Gamma'),\rho\sigma'(\Omega_1)\vdash\rho\sigma'(\Lambda_1),\rho\sigma'(\Delta),\rho\sigma'(\Delta'),\rho\sigma'(\delta_n[r])}\,\mathsf{Para}$$

(Completeness) Let us prove that $T_{(\mathcal{C},\sigma),\varphi} \subseteq |C_\psi(T_{(\mathcal{C},\sigma),\varphi})|$.

By definition, it is sufficient to prove that $T_{(\{\psi\},\sigma),\varphi} \subseteq \bigcup\limits_{(c,\sigma')\in Unf(\psi)} T_{(c,\sigma'\circ\sigma),\varphi}$. We then have to prove that for each ground substitution $\rho : V \to T_\Sigma$ such that $Sp \models \rho(\psi)$, there exists $(c,\sigma') \in Unf(\psi)$ such that there exists $\rho' : V \to T_\Sigma$ such that $Sp \models \rho'(\chi)$ for each $\chi \in c$. In other words, we have to prove that $\rho(\psi)$ can be deduced from specification *Sp* if there exists $(c,\sigma') \in Unf(\psi)$, and $\rho' : V \to T_\Sigma$ such that $Sp \models \rho'(\chi)$ for each $\chi \in c$.

If $\rho(\psi)$ is a tautology, it is a ground instance of a tautology $\sigma'(\psi)$. Therefore, there exists a pair $(\emptyset,\sigma') \in Unf(\psi)$ build by the **(Reduce)** part of the algorithm.

If it is not a tautology, let show that there exists a pair $(c,\sigma') \in Unf(\psi)$ and a substitution ρ' such that $\rho(\psi)$ can be deduced from the set of all $\rho'(\chi)$ for $\chi \in c$.

First, let us note that the unfolding procedure defines a strategy which bounds the search space for proof trees to a class of trees having a specific structure. The procedure defines a proof search strategy which selects proof trees where:

- no instance of cut and paramodulation occurs over instances of substitution
- no instance of cut occurs in the left-hand side subtree of an instance of paramodulation
- there is no instance of paramodulation whose premises both are instances of paramodulation
- there is no instance of cut whose premises both are instances of cut.

We then have to prove that there exists a proof tree having the structure we just described and of conclusion $\rho(\psi)$. We are actually going to prove a stronger result: we are going to define elementary transformations of proof trees, which allow to rewrite elementary combinations of inference rules, and then we will prove that the resulting global proof trees transformation is weakly normalizing and normal forms are proof trees with the above structure.

We give here the elementary transformations of basic proof trees. For instance, when an instance of the cut rule occurs over an instance of the substitution rule, we have the following rewriting rule to make the instance of substitution go over the instance of cut:

$$\dfrac{\dfrac{\Gamma\vdash\Delta,\varphi \quad \Gamma',\varphi\vdash\Delta'}{\Gamma,\Gamma'\vdash\Delta,\Delta'}\,\mathsf{Cut}}{\sigma(\Gamma),\sigma(\Gamma')\vdash\sigma(\Delta),\sigma(\Delta')}\,\mathsf{Subs} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma\vdash\Delta,\varphi}{\sigma(\Gamma)\vdash\sigma(\Delta),\sigma(\varphi)}\,\mathsf{Subs} \quad \dfrac{\Gamma',\varphi\vdash\Delta'}{\sigma(\Gamma'),\sigma(\varphi)\vdash\sigma(\Delta')}\,\mathsf{Subs}}{\sigma(\Gamma),\sigma(\Gamma')\vdash\sigma(\Delta),\sigma(\Delta')}\,\mathsf{Cut}$$

The case where an instance of substitution occurs over an instance of paramodulation:

$$\dfrac{\dfrac{\Gamma\vdash\Delta,s=t \qquad \Gamma'\vdash\Delta',\varphi[r]}{\sigma(\Gamma),\sigma(\Gamma')\vdash\sigma(\Delta),\sigma(\Delta'),\sigma(\varphi[t])}\,\mathsf{Para}}{\rho(\sigma(\Gamma)),\rho(\sigma(\Gamma'))\vdash\rho(\sigma(\Delta)),\rho(\sigma(\Delta')),\rho(\sigma(\varphi[t]))}\,\mathsf{Subs} \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\Gamma \vdash \Delta, s = t}{\rho(\sigma(\Gamma)) \vdash \rho(\sigma(\Delta)), \rho(\sigma(s)) = \rho(\sigma(t))}\,\mathsf{Subs} \qquad \dfrac{\Gamma' \vdash \Delta', \varphi[r]}{\rho(\sigma(\Gamma') \vdash \rho(\sigma(\Delta'), \rho(\sigma(\varphi[r]))}\,\mathsf{Subs}}{\rho(\sigma(\Gamma)), \rho(\sigma(\Gamma')) \vdash \rho(\sigma(\Delta)), \rho(\sigma(\Delta')), \rho(\sigma(\varphi[t]))}\,\mathsf{Para}$$

We know from the paramodulation in the initial proof tree that $\sigma(r) = \sigma(s)$, therefore we also have $\rho(\sigma(r)) = \rho(\sigma(s))$.

The case where an instance of cut occurs in the left-hand side of an instance of paramodulation:[6]

$$\dfrac{\dfrac{\vdash \psi \quad \psi \vdash s = t}{\vdash s = t}\,\mathsf{Cut} \qquad \vdash \varphi[r]}{\vdash \sigma(\varphi[t])}\,\mathsf{Para} \quad \rightsquigarrow \quad \dfrac{\dfrac{\psi \vdash s = t \quad \vdash \varphi[r]}{\sigma(\psi) \vdash \sigma(\varphi[t])}\,\mathsf{Para} \quad \dfrac{\vdash \psi}{\vdash \sigma(\psi)}\,\mathsf{Subs}}{\vdash \sigma(\varphi[t])}\,\mathsf{Cut}$$

The case where two instances of paramodulation occur over an instance of paramodulation:[7]

$$\dfrac{\dfrac{\vdash u = v \qquad \vdash s = t[x]}{\vdash \sigma(s) = \sigma(t[v])}\,\mathsf{Para} \qquad \dfrac{\vdash r = z \qquad \vdash \varphi[w]}{\vdash \sigma'(\varphi[z])}\,\mathsf{Para}}{\vdash \sigma''(\sigma'(\varphi[\sigma(t[v])]))}\,\mathsf{Para} \quad \rightsquigarrow$$

$$\dfrac{\dfrac{\dfrac{\vdash r = z}{\vdash \sigma'(r) = \sigma'(z)}\,\mathsf{Subs} \quad \dfrac{\vdash u = v \qquad \vdash s = t[x]}{\vdash \sigma(s) = \sigma(t[v])}\,\mathsf{Para}}{\vdash \sigma''(\sigma'(r)) = \sigma''(\sigma(t[v]))}\,\mathsf{Para} \qquad \dfrac{\vdash \varphi[w]}{\vdash \sigma'(\varphi[w])}\,\mathsf{Subs}}{\vdash \sigma''(\sigma'(\varphi[\sigma(t[v])]))}\,\mathsf{Para}$$

From the three instances of paramodulation in the initial tree, we know that: (1) $\sigma(x) = \sigma(u)$, (2) $\sigma'(w) = \sigma'(r)$, (3) $\sigma''(\sigma'(z)) = \sigma''(\sigma(s))$. In the resulting proof tree, we use these unifications in the order (1), (3) and (2).

The case of two cuts over a third one has to be divided into four cases, according to the position of the last cut formula in the premises of the two cuts of the top.

The case where φ is in both left premises:

$$\dfrac{\dfrac{\Gamma_1 \vdash \Delta_1, \varphi_1, \varphi \quad \Gamma_1', \varphi_1 \vdash \Delta_1'}{\Gamma_1, \Gamma_1' \vdash \Delta_1, \Delta_1', \varphi}\,\mathsf{Cut} \qquad \dfrac{\Gamma_2, \varphi \vdash \Delta_2, \varphi_2 \quad \Gamma_2', \varphi_2 \vdash \Delta_2'}{\Gamma_2, \Gamma_2', \varphi \vdash \Delta_2, \Delta_2'}\,\mathsf{Cut}}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \vdash \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\,\mathsf{Cut}$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \vdash \Delta_1, \varphi_1, \varphi \quad \Gamma_2, \varphi \vdash \Delta_2, \varphi_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, \varphi_1, \varphi_2}\,\mathsf{Cut} \qquad \Gamma_2', \varphi_2 \vdash \Delta_2'}{\Gamma_1, \Gamma_2, \Gamma_2' \vdash \Delta_1, \Delta_2, \Delta_2', \varphi_1}\,\mathsf{Cut} \qquad \Gamma_1', \varphi_1 \vdash \Delta_1'}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \vdash \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\,\mathsf{Cut}$$

The case where φ is in both right premises:

[6] We omit the contexts for the sake of readability.

[7] We omit the contexts for the sake of readability.

$$\dfrac{\dfrac{\Gamma_1\vdash\Delta_1,\varphi_1 \quad \Gamma_1',\varphi_1\vdash\Delta_1',\varphi}{\Gamma_1,\Gamma_1'\vdash\Delta_1,\Delta_1',\varphi}\,\mathsf{Cut} \qquad \dfrac{\Gamma_2\vdash\Delta_2,\varphi_2 \quad \Gamma_2',\varphi_2,\varphi\vdash\Delta_2'}{\Gamma_2,\Gamma_2',\varphi\vdash\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1,\Gamma_1',\Gamma_2,\Gamma_2'\vdash\Delta_1,\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}$$

$$\rightsquigarrow \dfrac{\Gamma_1\vdash\Delta_1,\varphi_1 \quad \dfrac{\Gamma_1',\varphi_1\vdash\Delta_1',\varphi \quad \dfrac{\Gamma_2\vdash\Delta_2,\varphi_2 \quad \Gamma_2',\varphi_2,\varphi\vdash\Delta_2'}{\Gamma_2,\Gamma_2',\varphi\vdash\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1',\Gamma_2,\Gamma_2',\varphi_1\vdash\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1,\Gamma_1',\Gamma_2,\Gamma_2'\vdash\Delta_1,\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}$$

The case where φ is in the left premise of the left cut, and in the right premise of the right cut:

$$\dfrac{\dfrac{\Gamma_1\vdash\Delta_1,\varphi_1,\varphi \quad \Gamma_1',\varphi_1\vdash\Delta_1'}{\Gamma_1,\Gamma_1'\vdash\Delta_1,\Delta_1',\varphi}\,\mathsf{Cut} \qquad \dfrac{\Gamma_2\vdash\Delta_2,\varphi_2 \quad \Gamma_2',\varphi_2,\varphi\vdash\Delta_2'}{\Gamma_2,\Gamma_2',\varphi\vdash\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1,\Gamma_1',\Gamma_2,\Gamma_2'\vdash\Delta_1,\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}$$

$$\rightsquigarrow \dfrac{\Gamma_2\vdash\Delta_2,\varphi_2 \quad \dfrac{\dfrac{\Gamma_1\vdash\Delta_1,\varphi_1,\varphi \quad \Gamma_1',\varphi_1\vdash\Delta_1'}{\Gamma_1,\Gamma_1'\vdash\Delta_1,\Delta_1',\varphi}\,\mathsf{Cut} \quad \Gamma_2',\varphi_2,\varphi\vdash\Delta_2'}{\Gamma_1,\Gamma_1',\Gamma_2',\varphi_2\vdash\Delta_1,\Delta_1',\Delta_2'}\,\mathsf{Cut}}{\Gamma_1,\Gamma_1',\Gamma_2,\Gamma_2'\vdash\Delta_1,\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}$$

The case where φ is in the right premise of the left cut, and in the left premise of the right cut:

$$\dfrac{\dfrac{\Gamma_1\vdash\Delta_1,\varphi_1 \quad \Gamma_1',\varphi_1\vdash\Delta_1',\varphi}{\Gamma_1,\Gamma_1'\vdash\Delta_1,\Delta_1',\varphi}\,\mathsf{Cut} \qquad \dfrac{\Gamma_2,\varphi\vdash\Delta_2,\varphi_2 \quad \Gamma_2',\varphi_2\vdash\Delta_2'}{\Gamma_2,\Gamma_2',\varphi\vdash\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1,\Gamma_1',\Gamma_2,\Gamma_2'\vdash\Delta_1,\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}$$

$$\rightsquigarrow \dfrac{\Gamma_1\vdash\Delta_1,\varphi_1 \quad \dfrac{\Gamma_1',\varphi_1\vdash\Delta_1',\varphi \quad \dfrac{\Gamma_2,\varphi\vdash\Delta_2,\varphi_2 \quad \Gamma_2',\varphi_2\vdash\Delta_2'}{\Gamma_2,\Gamma_2',\varphi\vdash\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1',\Gamma_2,\Gamma_2',\varphi_1\vdash\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}}{\Gamma_1,\Gamma_1',\Gamma_2,\Gamma_2'\vdash\Delta_1,\Delta_1',\Delta_2,\Delta_2'}\,\mathsf{Cut}$$

Let us define the measure of a proof tree π by:

$$m\left(\dfrac{\pi_1 \quad \pi_2}{\Gamma_1,\Gamma_2\vdash\Delta_1,\Delta_2}\mathsf{Cut}\right) = \begin{cases} 1+m(\pi_1)+m(\pi_2) & \text{if each } \pi_i = \dfrac{\pi_{i_1} \quad \pi_{i_2}}{\Gamma_i\vdash\Delta_i}\mathsf{Cut}\ i=1,2 \\ m(\pi_1)+m(\pi_2) & \text{otherwise} \end{cases}$$

$$m\left(\dfrac{\pi_1 \quad \pi_2}{\Gamma_1,\Gamma_2\vdash\Delta_1,\Delta_2}\mathsf{Para}\right) = \begin{cases} 1+m(\pi_1) & \text{if } \pi_1 = \dfrac{\pi_{1_1} \quad \pi_{1_2}}{\Gamma_1\vdash\Delta_1}\mathsf{Cut} \\ 1+m(\pi_1)+m(\pi_2) & \text{if each } \pi_i = \dfrac{\pi_{i_1} \quad \pi_{i_2}}{\Gamma_i\vdash\Delta_i}\mathsf{Para}\ i=1,2 \\ m(\pi_1)+m(\pi_2) & \text{otherwise} \end{cases}$$

A proof tree π is in *normal form* if and only if $m(\pi)=0$. A proof tree $\pi = \dfrac{\pi_1 \quad \pi_2}{\varphi}$ is called *maximal* if and only if π_1 and π_2 are in normal form but π is not. Therefore, by applying the strategy which consists in reducing first maximal proof trees, we show that the measure m decreases for each basic transformation given above.

Since by hypothesis, $Sp \models \rho(\psi)$, and ψ is not a tautology, there necessarily exists either an axiom

$$\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k$$

or an axiom

$$\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \vdash \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k, s = t$$

and a ground substitution ρ' such that

- $\rho'(\psi_i) = \rho'(\gamma_i)$ for each $1 \leq i \leq p$
- $\rho'(\varphi_i) = \rho'(\delta_i)$ for each $1 \leq i \leq q$
- $\rho'(\delta_{n_{|\omega}}) = \rho'(t)$.

Hence ρ' is a unifier of each ψ_i and γ_i, of each φ_i and δ_i and of $\delta_{n_{|\omega}}$ and t. So there exists a proof tree resulting of the transformation defined above, of conclusion $\rho(\psi)$ where $\rho = \rho'$, and of the form:

$$\dfrac{\dfrac{\vdots}{\rho(\Gamma') \vdash \rho(\xi_l), \rho(\Delta')} \quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \rho(\xi_2)} \quad \dfrac{\dfrac{\vdots}{\vdash \rho(\xi_1)} \quad ST}{\rho(\Gamma), \rho(\Omega_2) \vdash \rho(\Delta)}\,\mathsf{Cut}}{\vdots}\,\mathsf{Cut}}{\rho(\Gamma), \rho(\Omega_l) \vdash \rho(\Delta)}\,\mathsf{Cut}}{\rho(\Gamma), \rho(\Gamma') \vdash \rho(\Delta), \rho(\Delta')}\,\mathsf{Cut}$$

where ST is the following subtree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \Omega_1 \vdash \Lambda_1, \Delta}\,\mathsf{Ax}}{\rho(\Gamma), \rho(\Omega_1) \vdash \rho(\Lambda_1), \rho(\Delta)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\rho(\zeta_1) \vdash}}{\rho(\Gamma), \rho(\Omega_1) \vdash \rho(\Lambda_2), \rho(\Delta)}\,\mathsf{Cut} \quad \dfrac{\vdots}{\rho(\zeta_2) \vdash}}{\vdots}\,\mathsf{Cut}}{\rho(\Gamma), \rho(\Omega_1) \vdash \rho(\Delta)}\,\mathsf{Cut}$$

or of the form:

$$\dfrac{\dfrac{\vdots}{\rho(\Gamma') \vdash \rho(\xi_l), \rho(\Delta')} \quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash \rho(\xi_2)} \quad \dfrac{\dfrac{\vdots}{\vdash \rho(\xi_1)} \quad ST}{\rho(\Gamma), \rho(\Omega_2) \vdash \rho(\Delta), \rho(\delta_n[r])}\,\mathsf{Cut}}{\vdots}\,\mathsf{Cut}}{\rho(\Gamma), \rho(\Omega_l) \vdash \rho(\Delta), \rho(\delta_n[r])}\,\mathsf{Cut}}{\rho(\Gamma), \rho(\Gamma') \vdash \rho(\Delta), \rho(\Delta'), \rho(\delta_n[r])}\,\mathsf{Cut}$$

where ST is the following subtree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\rho(t)=\rho(r) \quad \dfrac{\mathsf{Subs}\dfrac{\dfrac{}{\Gamma,\Omega_1\vdash\Lambda_1,\Delta,s=t}\,\mathsf{Ax}}{\rho(\Gamma),\rho(\Omega_1)\vdash\rho(\Lambda_1),\rho(\Delta),\rho(s)=\rho(t)} \quad \dfrac{\vdots}{\rho(\Gamma')\vdash\rho(\Delta'),\rho(\delta_n[s])}}{\rho(\Gamma),\rho(\Omega_1),\rho(\Gamma')\vdash\rho(\Lambda_1),\rho(\Delta),\rho(\Delta'),\rho(\delta_n[t])}\,\mathsf{Para}}{\rho(\Gamma),\rho(\Omega_1),\rho(\Gamma')\vdash\rho(\Lambda_1),\rho(\Delta),\rho(\Delta'),\rho(\delta_n[r])}\,\mathsf{Para} \quad \dfrac{\vdots}{\rho(\zeta_1)\vdash}}{\rho(\Gamma),\rho(\Omega_1)\vdash\rho(\Lambda_2),\rho(\Delta),\rho(\delta_n[r])}\,\mathsf{Cut} \quad \dfrac{\vdots}{\rho(\zeta_2)\vdash}}{\vdots}\,\mathsf{Cut}}{\rho(\Gamma),\rho(\Omega_1)\vdash\rho(\Delta),\rho(\delta_n[r])}\,\mathsf{Cut}$$

and where $\Gamma = \{\psi_1, \ldots, \psi_p\}$, $\Delta = \{\varphi_1, \ldots, \varphi_q\}$, $\Gamma' = \{\gamma_{p+1}, \ldots, \gamma_m\}$, $\Delta' = \{\delta_{q+1}, \ldots, \delta_n\}$, for each i, $1 \leq i \leq l$, $\Omega_i = \{\xi_i, \ldots, \xi_l\}$ and for each i, $1 \leq i \leq k$, $\Lambda_i = \{\zeta_i, \ldots, \zeta_k\}$.

Conclusion

In this paper, we extended the test selection method known as axiom unfolding to quantifier-free first-order specifications with equality. We first proved the existence of an exhaustive test set, namely the set of observable semantic consequences of the specification, without any condition on the system under test. As it has originally been defined in the algebraic specifications setting, our unfolding procedure consists in dividing the initial test set for a formula into test subsets, characterising the different instances of this formula that can be proved from the specification axioms. The generation of a test set for this formula then arises from the selection of one test case in each resulting subset. We improved our previous works on this procedure by handling the equality predicate in an efficient way, thanks to the paramodulation rule. We proved this procedure to be sound and complete, so that exhaustivity for a given formula is preserved at each step. We finally proposed a strategy to cover the exhaustive test set.

The definition of test selection criteria is the first step towards the construction of a practical test set to submit to the system. The next step is the generation of a test set satisfying these criteria. In our framework, the generation consists in applying the uniformity hypothesis to the constrained test sets obtained by unfolding an initial test purpose. It actually comes down to solve the constraints associated to each constrained test purpose, in order to build one test case corresponding to this purpose. Note that for a constrained test set $T_{(\mathcal{C},\sigma),\varphi}$ where $\mathcal{C}$ is empty, any instantiation of the variables defines a test case. However, since we are interested in generating only one test case per constrained test set, a complete unfolding procedure is useless. It suffices to solve the constraints on the fly, during the unfolding procedure. A strategy is then needed to choose at each step which constraint in the current set $\mathcal{C}$ should be unfolded in order to reduce the remaining set $\mathcal{C}$ to the empty set and then to be able to compute a test case. Obviously, as the current set of constraints may be unsatisfiable, the test case generation also requires some backtracking steps. To be efficient, such a strategy of test case generation should be based on some heuristics (e.g. the choice of the next constraint to be unfolded may be based on syntactic criteria, like constraints with the fewest variables or constraints unifying with the fewest axioms) or applied to specifications written according to some predefined restricted form [25, 1]. To complete the work of this paper, we plan to study the definition of an efficient algorithm of test case generation for a subclass of quantifier-free first-order specification.

Ongoing research is also to be continued on structuration aspects. Axiomatic specifications can be structured via operators allowing to rename, to reduce or to enrich a specification for instance, or to make the union of two specifications. Integration testing, that is testing the composition of several modules or program units, deals with testing the new properties arising from the composition of those modules but that did not exist in the individual modules. Our goal then is to propose a framework of functional testing with selection criteria including structuration primitives, following for instance [23, 24].

References

1. Marc Aiguier, Agnès Arnould, Clément Boin, Pascale Le Gall, and Bruno Marre. Testing from algebraic specifications: test data set selection by unfolding axioms. In *Formal Approaches to Testing of Software (FATES'05)*, volume 3997 of *Lecture Notes in Computer Science*, pages 203–217, 2005.

2. Marc Aiguier, Agnès Arnould, Pascale Le Gall, and Delphine Longuet. Test selection criteria from quantifier-free first-order specifications. In *Fundamentals of Software Engineering (FSEN'07)*, volume 4767 of *Lecture Notes in Computer Science*, pages 144–159, 2007.
3. Marc Aiguier, Agnès Arnould, Pascale Le Gall, and Delphine Longuet. Exhaustive test sets for algebraic specification correctness. Technical report, IBISC, Université d'Évry Val d'Essonne, 2008. Submitted, available at http://www.epigenomique.genopole.fr/∼aiguier/publications/communications/exhaust.pdf.
4. Agnès Arnould and Pascale Le Gall. Test de conformité : une approche algébrique. *Technique et Science Informatiques, Test de logiciel*, 21:1219–1242, 2002.
5. Agnès Arnould, Pascale Le Gall, and Bruno Marre. Dynamic testing from bounded data type specifications. In *Dependable Computing - EDCC-2*, volume 1150 of *Lecture Notes in Computer Science*, pages 285–302, 1996.
6. Gilles Bernot. Testing against formal specifications: a theoretical view. In *Theory and Practice of Software Development (TAPSOFT'91)*, volume 494 of *Lecture Notes in Computer Science*, pages 99–119, 1991.
7. Gilles Bernot, Marie-Claude Gaudel, and Bruno Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387–405, 1991.
8. Achim D. Brucker and Burkhart Wolff. Symbolic test case generation for primitive recursive functions. In *Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, pages 16–32, 2004.
9. Matthieu Carlier and Catherine Dubois. Functional testing in the Focal environment. In *Tests and Proofs (TAP'08)*, volume 4966 of *Lecture Notes in Computer Science*, pages 84–98, 2008.
10. Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. *ACM SIGPLAN Notices*, 35(9):268–279, 2000.
11. Jeremy Dick and Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Formal Methods Europe (FME'93)*, volume 670 of *Lecture Notes in Computer Science*, pages 268–284, 1993.
12. Lars Frantzen, Jan Tretmans, and Tim A. C. Willemse. Test generation based on symbolic specifications. In *Formal Approaches to Software Testing (FATES'04)*, volume 3395 of *Lecture Notes in Computer Science*, pages 1–15, 2004.
13. Christophe Gaston, Pascale Le Gall, Nicolas Rapin, and Assia Touil. Symbolic execution techniques for test purpose definition. In *Testing of Communicating Systems (TestCom'06)*, volume 3964 of *Lecture Notes in Computer Science*, pages 1–18, 2006.
14. Marie-Claude Gaudel. Testing can be formal, too. In *Theory and Practice of Software Development (TASPOFT'95)*, volume 915 of *Lecture Notes in Computer Science*, pages 82–96, 1995.
15. Marie-Claude Gaudel and Pascale Le Gall. Testing data types implementations from algebraic specifications. In Robert M. Hierons, Jonathan P. Bowen, and Mark Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 209–239. Springer, 2008.
16. Rolf Hennicker, Martin Wirsing, and Michel Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2):393–443, 1997.
17. Claude Jard and Thierry Jéron. TGV: theory, principles and algorithms. a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Software Tools for Technology Transfer (STTT)*, 6, 2004.
18. Pieter W. M. Koopman, Artem Alimarine, Jan Tretmans, and Marinus J. Plasmeijer. GAST: Generic automated software testing. In *Implementation of Functional Languages*, volume 2670 of *Lecture Notes in Computer Science*, pages 84–100, 2002.
19. Pascale Le Gall and Agnès Arnould. Formal specification and test: correctness and oracle. In *11th Workshop on Algebraic Development Techniques (WADT'96)*, volume 1130 of *Lecture Notes in Computer Science*, pages 342–358, 1996.
20. David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.
21. Guillaume Lussier and Hélène Waeselynck. Proof-guided test: an experimental study. In *International Computer Software and Applications Conference (COMPSAC'04)*, pages 528–533. IEEE Computer Society, 2004.
22. Patrícia Machado. On oracles for interpreting test results against algebraic specifications. In *Algebraic Methodology and Software Technology*, volume 1548 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

23. Patrícia Machado. Testing from structured algebraic specifications. In *Algebraic Methodology and Software Technology (AMAST'00)*, volume 1816 of *Lecture Notes in Computer Science*, pages 529–544, 2000.
24. Patrícia Machado and Donald Sannella. Unit testing for CASL architectural specifications. In *Mathematical Foundations of Computer Science*, volume 2420 of *Lecture Notes in Computer Science*, pages 506–518, 2002.
25. Bruno Marre. LOFT : a tool for assisting selection of test data sets from algebraic specifications. In *Theory and Practice of Software Development (TAPSOFT'95)*, volume 915 of *Lecture Notes in Computer Science*, pages 799–800, 1995.
26. Peter D. Mosses. *CASL Reference Manual, The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004.
27. Alexandre Petrenko, Sergiy Boroday, and Roland Groz. Confirming configurations in EFSM. In *Formal Methods for Protocol Engineering and Distributed Systems (FORTE'99)*, volume 156 of *IFIP Conference Proceedings*, pages 5–24, 1999.
28. Marc Phalippou. *Relations d'implantation et hypothèses de test sur les automates à entrées et sorties*. PhD thesis, Université de Bordeaux, 1994.
29. George A. Robinson and Larry Wos. Paramodulation and theorem proving in first-order theories with equality. *Machine Intelligence*, 4:133–150, 1969.
30. Vlad Rusu, Lydie du Bousquet, and Thierry Jéron. An approach to symbolic test generation. In *2nd International Workshop on Integrated Formal Method (IFM'00)*, volume 1945 of *Lecture Notes in Computer Science*, pages 338–357, 2000.
31. Jan Tretmans. Testing labelled transition systems with inputs and outputs. In *International Workshop on Protocols Test Systems (IWPTS'95)*, 1995.