

Testing from Algebraic Specifications: Test Data Set Selection by Unfolding Axioms

Marc Aiguier¹, Agnès Arnould², Clément Boin¹, Pascale Le Gall¹
and Bruno Marre³

¹Université d'Évry-Val d'Essonne, LaMI CNRS UMR 8042,
523 pl. des Terrasses F-91025 Évry Cedex, France
{aiguier,cboin,legall}@lami.univ-evry.fr

²Université de Poitiers, SIC, CNRS FRE 2731,
SP2MI, F-86962 Futuroscope Cedex
arnould@sic.sp2mi.univ-poitiers.fr

³CEA/DRT/LIST/DTSI/SLA Saclay
F-91191 Gif sur Yvette Cedex
bruno.marre@cea.fr

Abstract. This paper deals with test data set selection from algebraic specifications. Test data set are generated from selection criteria which are usually defined to cover specification axioms. The unfolding selection criterion consists in covering the input domain of an operation using case analysis. The unfolding procedure can be iterated in order to split input domains of operations into finer subdomains. In this paper we propose to extend an unfolding procedure previously developed in [6, 22]. This yields a generic extension which can be applied to any positive conditional specification with constructors.

Keywords: Specification-based testing, algebraic specifications, selection criteria, unfolding, proof tree normalization, conditional rewriting.

1 Introduction

Specification-based testing, or black-box testing, consists in the dynamic verification of the specification requirements. Moreover, formal specifications are of great help for this task since it allows the design of well-founded and powerful tools for test case generation and for test execution. Test cases are then automatically generated from selection criteria. These criteria are chosen by experts according to either the application domain or the criticality level. Generally, criteria for specification-based testing allow to cover the specification requirements (e.g. axioms, transitions or states). In order to provide a success/failure verdict (oracle problem), test execution tools apply test inputs and analyse the outputs by comparison with the expected results defined from the formal specification.

Several approaches have been proposed, each one depending on the choice of formalisms: labelled transition systems [18], model based specifications such as B method, VDM or Z [12, 19], synchronous reactive languages as LUSTRE [23], algebraic specifications [6, 14, 7, 2, 3, 17, 20, 21, 13, 10, 9]. In the framework



of testing from algebraic specifications, decision procedures interpret test outputs such that the resulting verdicts fit on the notion of program correctness. Comparing the test outputs with the expected results may be a complex task when some information is missing (the oracle problem). Different observational approaches [8] have been proposed to cope with similar problems arising with specification refinement. Previous works [6, 14] and more particularly [5, 17, 2] provide a formal framework for a pure black-box testing from algebraic specifications. Test cases are observable formulas which can be computed by the program under test and interpreted as “true” or “false”. Correctness of a program under test with respect to a specification is then defined up to some observational equivalence depending on the set of observable formulas. Obviously, a correct program is necessarily successful for all test cases. On the contrary, if the success of a test set ensures the program correctness, then the test set is said to be exhaustive. Since under some minimal hypotheses on the program under test, an exhaustive test set can be viewed as a correctness reference for dynamic testing, it is a good candidate to be chosen as a starting test set for the selection step.

In this paper, we are interested in the process of selecting test sets from algebraic specifications. Within this framework, a selection criterion has to be viewed as the coverage of some formulas which represent some test objectives, such as the axioms of the specifications. There are two main strategies to select test cases: one that performs any selection of test cases based on some deterministic choice or on a distribution on the considered input domain (random testing) and one that performs a selection of test cases in order to cover subdomains identified by a domain coverage (partition testing). In the latter case, subdomains partition the initial domain and correspond to the various cases addressed by the specification. Concerning random testing, it has been advocated by several works [7, 10] since either it is really easy to implement or it brings a quantitative evaluation of the testing process. The widely well-known drawback of random testing is the case of a subdomain with a low probability level but with a high probability level of failure rate. Within the framework of testing from algebraic specifications, such an unlikely subdomain arises with conditional axioms of the form $\varphi(X) \Rightarrow \psi(X)$ with X a variable vector. If the subdomain making true the condition $\varphi(X)$ has a low probability to be drawn, then random testing can miss the verification of $\psi(X)$ which is precisely required on this problematic subdomain [6, 9]. On the contrary, partition testing is based on a case analysis of the formula under test. The formula under test is preprocessed in order to reveal pertinent subdomains. For example, [12] translates formula under test into an equivalent disjunctive normal form, each conjunction representing a test subdomain. Another formula translation consists in applying a proof strategy such that the remaining lemmas represent a test subdomain [9]. [6, 22] have given importance to case analysis by unfolding specification axioms. It consists in splitting the input domain of an operation from specification axioms. Selection criteria based on axiom unfolding allow the tester to progressively refine the coverage domain in order to control the size of the resulting test set.

The paper is organized as follows. In Section 2, we recall standard nota-

tions about algebraic positive conditional specifications. In order to be as self-contained as possible, Section 3 gives relevant definitions of [17]. In Section 4, we recall the previous unfolding procedure defined for a restricted class of conditional conditional specifications, the executable ones for which each computation has a unique normal form. Section 5 introduces an extension of this unfolding procedure allowing us to define a selection criterion for the class of all positive conditional specifications. Both unfolding procedures perform a case analysis on specification axioms defining the operations. We will show that the unfolding selection criterion performs at each step an adequate partition of the input domain insofar as it is a sound (no test is added) and complete (no test is lost) selection criterion.

2 Preliminaries

An *(algebraic) signature* $\Sigma = (S, F, V)$ consists in a set S of sorts, a set F of function names each one equipped with an arity in $S^* \times S$ and a S -indexed sets of variables V . In the sequel, a function f with the arity $(s_1 \dots s_n, s)$ will be noted $f : s_1 \times \dots \times s_n \rightarrow s$. A *signature with constructors* is a signature $\Sigma = (S, F, V)$ such that F has a subset C elements of which are called *constructors*. We note $\Omega = (S, C, V)$ the restriction of Σ to constructors of C . Given a signature $\Sigma = (S, F, V)$, $T_\Sigma(V)$ and T_Σ are both S -sets of *terms with variables in V* and *ground terms*, respectively, freely generated from variables and functions in Σ and preserving arity of functions. Using a standard numbering of the tree nodes by natural number strings, we can refer to positions in a term. Thus, given a term t , a *position* of t is a string ω in \mathbb{N} which represents the path from the root of t to the subterm whose the head function occurs at this position. This subterm is noted $t|_\omega$. Given a position $\omega \in \mathbb{N}^*$ in a term t , $t[t']_\omega$ is the term obtained from t by substituting the subterm $t|_\omega$ by t' . A *substitution* is any mapping $\rho : V \rightarrow T_\Sigma(V)$ that preserves sorts. They are naturally extended to terms with variables. Σ -*equations* are formulae of the form $t = t'$ with $t, t' \in T_\Sigma(V)_s$ for $s \in S$. A *positive conditional Σ -formula* is then any sentence of the form $\alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \alpha_{n+1}$ where each α_i is a Σ -equation ($1 \leq i \leq n + 1$). $Sen(\Sigma)$ is the set of all positive conditional Σ -formulae. Given a formula $\varphi \in Sen(\Sigma)$, $Var(\varphi)$ is the set of all variables occurring in φ . A *(positive conditional) specification* $SP = (\Sigma, Ax)$ consists in a signature Σ and a set Ax of positive conditional formulae often called *axioms*.

A Σ -*algebra* \mathcal{A} is a S -indexed set A equipped for each $f : s_1 \times \dots \times s_n \rightarrow s \in F$ with a mapping $f^{\mathcal{A}} : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$. A Σ -morphism μ from a Σ -algebra \mathcal{A} to a Σ -algebra \mathcal{B} is a mapping $\mu : A \rightarrow B$ such that for all $s \in S$, $\mu(A_s) \subseteq B_s$ and for all $f : s_1 \times \dots \times s_n \rightarrow s \in F$ and all $(a_1, \dots, a_n) \in A_{s_1} \times \dots \times A_{s_n}$ $\mu(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(\mu(a_1), \dots, \mu(a_n))$. $Alg(\Sigma)$ is the category objects of which are all Σ -algebras. The set of ground terms T_Σ can be extended into a Σ -algebra by providing each function name $f : s_1 \times \dots \times s_n \rightarrow s \in F$ with an application $f^{T_\Sigma} : (t_1, \dots, t_n) \mapsto f(t_1, \dots, t_n)$. Given a Σ -algebra \mathcal{A} , we note $\underline{\cdot}^{\mathcal{A}} : T_\Sigma \rightarrow \mathcal{A}$ the unique Σ -morphism that maps any $f(t_1, \dots, t_n)$ to

$f^A(t_1^A, \dots, t_n^A)$. A Σ -algebra \mathcal{A} is said *reachable* if $_A$ is surjective. $Gen(\Sigma)$ is the full subcategory of $Alg(\Sigma)$ objects of which are all reachable Σ -algebras. Given a Σ -algebra \mathcal{A} , a Σ -interpretation in \mathcal{A} is any mapping $\iota : V \rightarrow \mathcal{A}$. They are naturally extended to terms with variables. A Σ -algebra \mathcal{A} *validates* a Σ -formula $\varphi : \bigwedge_{1 \leq i \leq n} t_i = t'_i \Rightarrow t = t'$, noted $\mathcal{A} \models \varphi$, if and only if for every Σ -interpretation ι in \mathcal{A} , if $\iota(t_i) = \iota(t'_i)$ then $\iota(t) = \iota(t')$. Given $\Psi \subseteq Sen(\Sigma)$ and two Σ -algebras \mathcal{A} and \mathcal{B} , \mathcal{A} is Ψ -*equivalent* to \mathcal{B} , noted $\mathcal{A} \equiv_\Psi \mathcal{B}$, if and only if we have: $\forall \varphi \in \Psi, \mathcal{A} \models \varphi \iff \mathcal{B} \models \varphi$. Given a specification $SP = (\Sigma, Ax)$, a Σ -algebra \mathcal{A} is a *SP-algebra* if for every $\varphi \in Ax$, $\mathcal{A} \models \varphi$. $Alg(SP)$ and $Gen(SP)$ are both full subcategories of $Alg(\Sigma)$ and $Gen(\Sigma)$, respectively, objects of which are all *SP-algebras*. A Σ -formula φ is a *semantical consequence* of a specification $SP = (\Sigma, Ax)$, noted $SP \models \varphi$, if and only if for every *SP-algebra* \mathcal{A} , we have $\mathcal{A} \models \varphi$. SP^\bullet is the set of all semantical consequences. A sound and complete calculus for positive conditional specifications is defined by the following inference rules:

$$\begin{array}{c}
\text{Ref.} \frac{}{SP \vdash t = t} \quad \text{Sym.} \frac{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = t'}{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t' = t} \\
\text{Trans.} \frac{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = t' \quad SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t' = t''}{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = t''} \\
\text{Context} \frac{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t_1 = t'_1 \dots SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t_n = t'_n}{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)} \\
\text{Axiom} \frac{\varphi \in Ax}{(\Sigma, Ax) \vdash \varphi} \quad \text{Subst.} \frac{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow \alpha}{SP \vdash \bigwedge_{1 \leq i \leq m} \sigma(\alpha_i) \Rightarrow \sigma(\alpha)} \\
\text{Monotony} \frac{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow \alpha}{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \wedge \beta \Rightarrow \alpha} \quad \text{M.P.} \frac{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \wedge u = v \Rightarrow \alpha \quad SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow u = v}{SP \vdash \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow \alpha}
\end{array}$$

3 Testing from algebraic specifications

The interpretation of test case submission as a success or failure is closely related to the notion of program correctness. More precisely, any test case submitted to a correct program should be analysed in a success case while ideally, an incorrect program should fail for at least a test case. In the context of testing from algebraic specifications, a first natural testing hypothesis is to suppose that programs are denoted by Σ -algebras. Hence, the test case interpretation can be defined from the satisfaction of some formulae. These formulae link input test data to expected results using operations, predicates or connectors of the specification. Thus, following our previous works [6, 17, 2], test cases are then denoted by formulae. As test case submission should yield a verdict, the formulae that represent test cases correspond to a subset of all formulae: the set of all formulae which can be interpreted by a computation of the program as “true” or “false”. These “executable” formulae are also called *observable*. In practice, observable formulae are ground formulae only involving equalities on some given sorts (for example, all sorts provided with an equality predicate within the programming language). Some abstract data types (sets, stacks, etc) do not provide an equality procedure within the program under test and then, have to be observed through successive applications of functions leading to an observable result. For example, we can use set cardinality and element membership to observe the set data type as well as the height and the top of all successive popped stacks for the stack data type. Thus, a non observable ground equality of the form $t = t'$ is often observed through all observable contexts $c[\cdot]$ applied to both t and t' . It amounts to apply to both terms t and t' the same successive application of operations yielding an observable value, and to compare resulting values.

3.1 A general framework

Let $SP = (\Sigma, Ax)$ be a positive conditional specification and $Obs \subseteq Sen(\Sigma)$ any set of observable formulae. Let P be a program which is assimilated to a Σ -algebra of $Alg(\Sigma)$. It is sensible to assume that all values used by P are denoted by operation composition of Σ and that P has a functional behaviour with respect to the operations of Σ . Actually, our notion of correctness is based on this hypothesis. Indeed, under this minimal hypothesis, the program under test can be viewed as a simple reachable Σ -algebra which evaluates terms as the way the program computes the observable formulae. Then, test cases are observable formulae, which are successful for the program under test if and only if the Σ -algebra P satisfies them (i.e. executes them and interpret them as “true”):

Definition 1 (Test case and test set). A *test case* is a formula of $SP^\bullet \cap Obs$. A *test set* T is a set of test cases. T is said to be *successful* for P if and only if $\forall \varphi \in T, P \models \varphi$.

Correctness for dynamic testing is defined following an observational approach comparable to the ones used to define refinement of specifications: it is

required that an algebra of the concrete specification is observationally equivalent to an algebra of the abstract specification. Here, by analogy, to be qualified as correct with respect to a specification, a program is required to be observationally equivalent to an algebra of the specification up to the observable formulae of Obs .

Definition 2 (Correctness). P is *correct* for SP via Obs , denoted by $Correct_{Obs}(P, SP)$, if and only if there exists an algebra \mathcal{A} in $Alg(SP)$ such that $\mathcal{A} \equiv_{Obs} P$.

Note that our definition of test cases guarantees that any correct program is necessarily successful for the set of all test cases $SP^\bullet \cap Obs$. Indeed, SP^\bullet is clearly the largest set of formulae which are both satisfied by all SP -algebras and executable by any program under test capable of interpreting formulae in Obs . This property is also called the unbiased property ([6]). When reciprocally, a successful test set ensures correctness of the program under test, then we say that the test set is exhaustive:

Definition 3 (Exhaustiveness). A test set T is *exhaustive* for SP via Obs if and only if

$$\forall P \in Alg(\Sigma), P \models T \Leftrightarrow Correct_{Obs}(P, SP)$$

The existence of an exhaustive test set *Exhaust* means that the considered specification SP is testable via Obs since correctness can be asymptotically approached by submitting a (possibly infinite) test set. However, depending on the nature of SP and Obs , such an exhaustive test set does not necessarily exist. In [17], there is a counter-example of a specification SP and a set Obs of observable formulae for which the largest test set $SP^\bullet \cap Obs$ is not exhaustive.

Test sets can be compared with respect to their ability to reject (or to accept from a dual point of view) programs: a test set T is more efficient than a test set T' , denoted by $T \leq T'$, if and only if for all programs P in $Alg(\Sigma)$, then $P \models T$ implies $P \models T'$. T and T' verifying both $T \leq T'$ and $T' \leq T$ are said *equivalent*. Since $SP^\bullet \cap Obs$ is the largest acceptable test set, then $SP^\bullet \cap Obs$ is equivalent to any test set exhaustive for SP via Obs .

The works [20, 21] have proposed a slightly different point of view to formally define testing from algebraic specifications. The authors propose an oracle procedure combining black-box and white-box testing. A non observable equality is interpreted using two approximate equalities, qualified as resp. sound and complete. In practice, the complete equality often stands for an observation of equalities through a finite set of observable contexts (black-box testing) while the sound equality stands for an observation of equalities using the concrete implemented equality (white-box testing). This approach may be applied with first order formulae with some restrictions about the use of universal and existential quantifiers. Depending on its place in the formulae, an occurrence of a given equality is observed through either the complete or the sound equality. Intuitively, equalities within positive (resp. negative) literals are observed through complete (resp. sound) equalities. Finally, testing each axiom of the specification

through approximate equalities guarantees that any SP -algebra successfully interprets all test cases. Moreover, the approach has been extended to structured or CASL architectural specifications. However, there is no counterpart to the unbiased property: there is no care to the correctness question: how far can a successful program be declared as correct with respect to the specification?

The challenge of testing consists then in managing (infinite) test sets. In practice, experts apply some selection criteria on a reference test set in order to extract a test set of size sufficiently reasonable to be submitted to the program. The underlying idea is that all test sets satisfying a considered selection criterion reveal the same class of incorrect programs, intuitively the ones corresponding to the fault model captured by the criterion. For example, the criterion called “uniformity hypothesis” postulates that any chosen value is equivalent to another one. For example, if a test set is given by $\{\sigma(\varphi) \mid \sigma : V \rightarrow T_\Sigma\}$ where φ denotes a formula (e.g. an axiom of SP) built over the variable x , then the uniformity selection criterion consists in choosing one arbitrary substitution $\sigma_0 : V \rightarrow T_\Sigma$ in order to select one test: $\sigma_0(\varphi)$.

Roughly, a selection criterion C splits a given starting test set T into a family of test subsets $\{T_i\}_{i \in I_{C(T)}}$ such that $T = \bigcup_{i \in I_{C(T)}} T_i$ holds. A test set

satisfying such a selection criterion simply contains at least a test case for each non empty subset T_i . Intuitively, all test cases in T_i are supposed equivalent to reveal incorrect programs with respect the fault model captured by T_i . In practice, T represents a property φ (an operation, an axiom or any formula chosen as an testing objective) to be partially covered by testing. The sets T_i then represent subproperties of φ . The selection criterion C is then a coverage criterion according to the way C is splitting the initial test set T into the family $\{T_i\}_{i \in I_{C(T)}}$. This is a rather classical way to select test data, known under the term of *partition testing*.

Definition 4 (Selection criterion). A *selection criterion* C is a mapping¹ $\mathcal{P}(SP^\bullet \cap Obs) \rightarrow \mathcal{P}(\mathcal{P}(SP^\bullet \cap Obs))$. For a test set T , we note $|C(T)| = \bigcup_{i \in I_{C(T)}} T_i$

where $C(T) = \{T_i\}_{i \in I_{C(T)}}$.

T' satisfies C applied to T , noted by $T' \sqsubseteq C(T)$ if and only if:

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

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

Definition 5 (Properties). Let C, C' be two selection criteria and T, T' two test sets.

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

- C is said *sound for T* if and only if $|C(T)| \subseteq T$
- C is said *complete for T* if and only if $|C(T)|$ and T are equivalent test sets.
- C is *partitionning T* if and only if $\forall i, j \in I_{C(T)}, i \neq j \Rightarrow T_i \cap T_j = \emptyset$
- C is said *finer than C'* , denoted by $C \leq C'$ if and only if

$$\forall T, T' \subseteq SP^\bullet \cap Obs, T' \sqsubset C(T) \Rightarrow T' \sqsubset C'(T).$$
- A family of selection criteria $\{C_k\}_{k \in \mathcal{C}}$ is said *iterative* if and only if

$$\forall k \in \mathcal{C}, \exists k' \in \mathcal{C}, C_{k'} \leq C_k.$$

The properties of soundness and completeness are essential for an adequate selection criterion: soundness ensures that test cases will be selected within the starting test set (i.e. no test is added) while completeness ensures that we capture all test cases up to the notion of equivalent test cases (i.e. no test is lost). A sufficient condition to ensure both soundness and completeness of a selection criterion C for a test set T consists in showing that $|C(T)| = T$. This condition will be used in Section ???. When C is partitionning T , this means that the different test sets T_i are not superposed, and then, that one should choose at least a different test case for each T_i to build a test set satisfying C for T . An iterative family of selection criteria allows the tester to extend and to precise the process of test selection up to get a test set of convenient size. In order to obtain such an iterative family of selection criteria, it suffices to nest selection criteria. More precisely, if a selection criterion C_k build a test set family $\{T_1, \dots, T_{n_k}\}$ for a given test set T , and if a generic selection criterion C applied to any test set of this family, say T_i for example, provides the family $\{T_i^1, \dots, T_i^{n_i}\}$, then we can consider the selection criterion $C_{k'}$ defined by $C_{k'}(T) = \{T_1, \dots, T_{i-1}, T_i^1, \dots, T_i^{n_i}, T_{i+1}, \dots, T_{n_k}\}$. If moreover the intermediate selection criterion C is sound and complete, then clearly $C_{k'}$ is finer than C_k . We can systematically apply the selection criterion C to any test set T_i occurring in $C_k(T)$ for arbitrary index k in \mathcal{C} and test set T . Of course, in practice, to define selection criteria in a generic and concise way, they will be defined on test sets given in an intentional definition. Section ??? will define such an iterative family of selection criteria based on a generic procedure, the unfolding one which makes a case analysis of each occurrence of non-constructor operation according to specification axioms.

3.2 A reference test set

In this section, we will show that for a family of positive conditional specifications $SP = (\Sigma, Ax)$, there is a reference exhaustive test set via $Obs = \{u = v \mid u, v \in T_\Sigma\}$ ². We restrict ourselves to equations for several reasons. First, it simplifies the activity of test submission and success/failure decision since in practice, it simply amounts to execute two terms and to compare their results. Second, only

² If the signature $\Sigma = (S, F, V)$ has a subset $S_{obs} \subset S$ to denote observable sorts, then $Obs = \{u = v \mid \exists s \in S_{obs}, u, v \in T_{\Sigma_s}\}$.

considering equations instead of positive conditional formula will facilitate the definition of testing strategies. Let us introduce the following test set:

Definition 6 (Reference test set). Let $SP = (\Sigma, Ax)$ be a specification where $\Sigma = (S, F, V)$ is a signature. Let us define the set $T_0(SP)$ as follows:

$$T_0(SP) = \{f(u_1, \dots, u_n) = v \mid f \in F, u_1, \dots, u_n, v \in T_\Sigma, SP \vdash f(u_1, \dots, u_n) = v\}$$

This set is a reference for the test activity because it reflects the practice of testing: an operation under testing is both applied and compared to input data and result that are naturally denoted by ground terms in T_Σ . Moreover, this test set is maximal with respect to the chosen set Obs . Indeed, it corresponds to $SP^\bullet \cap Obs$.

Elements in $T_0(SP)$ are too numerous (often infinite) to be manageable. A subset of $T_0(SP)$ with a manageable size has to be selected.

For many test methods (anyway all test methods used in practice), some strategy schemata are proposed to guide test selection. The selection method that we define in this section takes inspiration from classic methods that partition (more generally that split into) the input domain of each function.

Let $\Sigma = (S, F, V)$ be a signature and f be an operation of Σ . Succinctly, our method reports under the following form:

1. splitting the input domain of f into many subdomains, called *test sets* for f , and
2. choosing any input in each non-empty sub-domain.

First, let us define what input domain and test set for signature operations are:

Definition 7 (Input domain of operations). Let $SP = (\Sigma, Ax)$ be a specification. Let $f : s_1 \times \dots \times s_n \rightarrow s$ be an operation of Σ . The *domain of f* , noted $T_0(SP)|_f$, is the set defines by:

$$T_0(SP)|_f = \{f(u_1, \dots, u_n) = v \mid f(u_1, \dots, u_n) = v \in T_0(SP)\}$$

Definition 8 (Test set for operations). Let $SP = (\Sigma, Ax)$ be a specification where $\Sigma = (S, F, V)$ is a signature. Let \mathcal{C} be a set of Σ -equations called *set of Σ -constraints*. Let $f : s_1 \times \dots \times s_n \rightarrow s$ be an operation of Σ .

A *test set for f with respect to \mathcal{C}* , noted $T_{\mathcal{C},f}$, is the set of ground equations defined by:

$$T_{\mathcal{C},f} = \{f(\sigma(x_1) \dots, \sigma(x_n)) = \sigma(y) \mid \sigma : V \rightarrow T_\Sigma, \forall \varepsilon \in \mathcal{C} SP \models \sigma(\varepsilon)\}$$

4 The selection criteria based on axiom unfolding in LOFT

In this section, we formalize the problem of test selection from algebraic specifications such as implemented in the test selection tool LOFT [6, 22].

4.1 The unfolding procedure in LOFT

The unfolding procedure as implemented in the test selection tool LOFT assumes that any conditional positive specification SP is actually presented under the form of a conditional rewrite system \mathcal{R} , that is any axiom is a conditional rewrite rule of the form $\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) \rightarrow v$ or can be directly transformed into

a conditional rewrite rule (e.g. by imposing that specifications presents graceful presentations [?]). Moreover, to ensure both soundness and completeness of the unfolding procedure with respect to the reference test set $T_0(SP)$, \mathcal{R} is assumed to be both confluent and equipped with a well-founded ordering satisfying for all $\bigwedge_{1 \leq i \leq m} t_i = t'_i \Rightarrow t \rightarrow t'$ in \mathcal{R} and all substitutions σ :

- $\sigma(t) > \sigma(t')$ and
- $\sigma(t) > \sigma(t_i)$ and $\sigma(t) > \sigma(t'_i)$ for all $1 \leq i \leq m$.

\mathcal{R} is then reductive [15]. It is well-known that such rewrite systems ensure that both join conditional rewriting is decidable [15] and: $u \xrightarrow{*} \mathcal{R} v \iff SP \vdash u = v$.

The unfolding procedure developed here, has in input:

- an operation $f \in F$,
- a conditional positive specification $SP = (\Sigma, Ax)$ presented under the form of a reductive and confluent rewrite system, and
- a set Γ of Σ -constraint sets.

The first set $\Gamma_0 = \{ \{ f(x_1, \dots, x_n) = y \} \}$ where $x_i, y \in V$ ($1 \leq i \leq n$).

The unfolding procedure is expressed by the two following inference rules:

Reduce

$$\frac{\Gamma \cup \{ \mathcal{C} \cup \{ t = t \} \}}{\Gamma \cup \{ \mathcal{C} \}}$$

Unfolding

$$\frac{\Gamma \cup \{ \mathcal{C} \cup \{ t = r \} \}}{\Gamma \cup \bigcup_{c \in Tr(u|_{\omega}, t=r)} \{ \mathcal{C} \cup c \}} \quad \omega \text{ a position in } u \text{ and } u \in \{ t, r \}$$

where $Tr(u|_\omega, t = r)$ for $t = r$ a Σ -equation not of the form $v = v$, is the set of Σ -constraint sets defined by:

$$Tr(u|_\omega, t = r) = \left\{ \left\{ u_1 = v_1, \dots, u_n = v_n, t[v]_\omega = r, \alpha_1, \dots, \alpha_m \right\} \left| \begin{array}{l} u|_\omega = g(u_1, \dots, u_n) \\ \sigma(u|_\omega) = \sigma(g(v_1, \dots, v_n)) \text{ (}\sigma \text{ unifier)}, \\ \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) \rightarrow v \in Ax \end{array} \right. \right\}$$

As the definition of $Tr(u|_\omega, t = r)$ is based on the subterm relation and unification, this set is computable if the specification SP has a finite set of axioms. Hence, given an equation $t = r$ we have the selection criterion C_ε that maps any $T_{\mathcal{C},f}$ to $\{T_{\mathcal{C} \setminus \{t=r\}} \cup c\}_{c \in Tr(u|_\omega, t=r)}$ if $t = r \in \mathcal{C}$, $T_{\mathcal{C},f}$ otherwise.

We write $\Gamma \vdash_U \Gamma'$ to indicate that Γ can be transformed to Γ' by applying one of the above inference rules.

Hence, an unfolding procedure is a program that accepts in input a positive conditional specification $SP = (\Sigma, Ax)$ and uses the above inference rules to generate a (finite or infinite) sequence

$$\Gamma_0 \vdash_U \Gamma_1 \vdash_U \Gamma_2 \vdash_U \Gamma_3 \vdash_U \dots$$

where $\Gamma_0 = \{\{f(x_1, \dots, x_n) = y\}\}$ with x_i, y ($1 \leq i \leq n$) are variables of Σ .

This unfolding procedure has been implemented in the test selection tool LOFT [6, 22] which has been developed in PROLOG. This enables it to benefit from a powerful resolution procedure of constraints. To illustrate this tool on an example, let us specify the *insert* operation with respect to the *List* constructors. This gives rise to the following specification written in the specification language CASL:

```

spec INSERT =
  NAT
then
  type List ::= [] | _::__(Nat; List)
  op insert : Nat × List → List
  ∀ x, y: Nat; L: List
  • insert(x, []) = x :: []                                %(insert_empty)%
  • x ≤ y ⇒ insert(x, y :: L) = x :: y :: L                %(insert_leq)%
  • ¬ x ≤ y ⇒ insert(x, y :: L) = y :: insert(x, L)        %(insert_g)%
end

```

It is obvious to transform this specification into an equivalent rewrite system by orienting each conclusion of axioms from the left to the right. With the recursive path ordering $>^{rpo}$ resulting from the precedence ordering: *insert* $>$ $_ :: _ > []$, this rewrite system is reductive. Therefore, let us use LOFT to split the domain of *insert* operation by unfolding its axioms. Hence, we obtain three selection constraints corresponding to the three axioms of *insert*. The following

LOFT command expresses that the \leq predicate must not be unfolded while the *insert* operation must be unfolded once.

```
??- unfold_std([#('___<__:Nat,Nat->boolean',0),#('insert:nat,list->list',1)],
insert(X,L1) = L2).
```

FINAL BINDING:

```
L1:list = empty
L2:list = ___:__(X:nat,empty)
SOLUTION #1,      CPUTIME = 0
```

FINAL BINDING:

```
L1:nlist = ___:__(_v0:nat,_v1:list)
L2:nlist = ___:__(X:nat,___:__(_v0,_v1))
REMAINING CONSTRAINTS = { ___<__(X,_v0) = true }
SOLUTION #2,      CPUTIME = 0
```

FINAL BINDING:

```
L1:nlist = ___:__(_v0:nat,_v1:nlist)
L2:nlist = ___:__(_v0,_v2:nlist)
REMAINING CONSTRAINTS = { ___<__(X:nat,_v0) = false, insert(X,_v1) = _v2 }
SOLUTION #3,      CPUTIME = 0
```

GLOBAL TIME ELAPSED = 0

NUMBER OF SOLUTIONS = 3

yes

Note that LOFT uses a purely equational logic. Consequently, predicates are translated as boolean operations. Now, if we unfold twice the *insert* operation, only the subdomain #3 is split, because only the third constraint contains an equation with an occurrence of *insert*. We do not give the LOFT outputs here, but only the test cases produced to cover the last subdomain.

SOLVED CONSTRAINTS:

BINDING:

```
L1:nlist = cons(_v0:nat,cons(_v1:nat,_v2:nlist))
L2:nlist = cons(_v0,cons(_v1,_v3:nlist))
CONSTRAINTS = { ___<__(X:nat,_v0) = false, ___<__(X,_v1) = false,
insert(X,_v2) = _v3 }
```

FINAL BINDING:

```
X:nat = 6
L1:nlist = ___:__(1,___:__(0,___:__(2,___:__(0,___:__(9,empty))))))
L2:nlist = ___:__(1,___:__(0,___:__(2,___:__(0,___:__(6,___:__(9,empty))))))
```

SOLUTION #5, CPUTIME = 9

4.2 Soundness and completeness

Test sets for operations are naturally extended to set of constraint sets as follows: Let Γ be a set of Σ -constraint sets and f be an operation of the signature Σ

$$T_{\Gamma,f} = \bigcup_{\mathcal{C} \in \Gamma} T_{\mathcal{C},f}$$

The completeness result needs to assume that for any Γ resulting of the unfolding procedure, any $\mathcal{C} \in \Gamma$, any $\varepsilon \in \mathcal{C}$ and any $\varphi \in Ax$, $Var(\varepsilon) \cap Var(\varphi) = \emptyset$. This can be easily obtained at each iteration of the unfolding procedure by renaming variables by fresh ones.

Therefore, for any specification SP presented under the form of a reductive and confluent rewrite system \mathcal{R} , both soundness and completeness of the unfolding procedure hold. Indeed, we have:

Theorem 9. *If $\Gamma \vdash_U \Gamma'$ then $T_{\Gamma,f} = T_{\Gamma',f}$.*

Proof. The case of the inference rule **Reduce** is obvious. The case of the inference rule **Unfolding** is proven as follows:

- (Soundness) $T_{\Gamma',f} \subseteq T_{\Gamma,f}$. By the definition of the inference rule, this amounts to show

$$\forall c \in Tr(u|_{\omega}, t = r), T_{c,f} \neq \emptyset \implies T_{c,f} \subseteq T_{\{t=r\},f}$$

for any $\mathcal{C} \in \Gamma$ and any $t = r \in \mathcal{C}$. Therefore, let $c \in Tr(u|_{\omega}, t = r)$ such that $T_{c,f} \neq \emptyset$. Without loss of generality, let us suppose that

$$c = \{u_1 = v_1, \dots, u_n = v_n, t[v]_{\omega} = r, \alpha_1, \dots, \alpha_m\}$$

with $t|_{\omega} = g(v_1, \dots, v_n)$, and $\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) \rightarrow v \in Ax$. More-

over, let $\sigma : V \rightarrow T_{\Sigma}$ be a ground substitution such that $SP \vdash \sigma(u_i) = \sigma(v_i)$, $SP \vdash \sigma(t[v]_{\omega}) = \sigma(r)$ and for all $1 \leq i \leq m$, $SP \vdash \sigma(\alpha_i)$. As SP presents a reductive and confluent rewrite system, we directly have $\sigma(t[g(v_1, \dots, v_n)]_{\omega}) \rightarrow \sigma(t[v]_{\omega})$. Moreover, by the confluence property, we have $\sigma(t[v]_{\omega}) \overset{*}{\leftrightarrow} \sigma(r)$, and then $\sigma(t[g(v_1, \dots, v_n)]_{\omega}) \overset{*}{\leftrightarrow} \sigma(r)$. Finally, we also have $\sigma(u_i) \overset{*}{\leftrightarrow} \sigma(v_i)$ and then $\sigma(g(u_1, \dots, u_n)) \overset{*}{\leftrightarrow} \sigma(g(v_1, \dots, v_n))$ whence $\sigma(t) \overset{*}{\leftrightarrow} \sigma(t[g(v_1, \dots, v_n)]_{\omega})$. We then conclude $\sigma(t) \overset{*}{\leftrightarrow} \sigma(r)$ and then $SP \vdash t = r$.

- (Completeness) To show the completeness, let us suppose that Γ has been transformed into Γ' from a constraint $t = r$ in Γ and a position ω in t . As the conditional rewrite system that represents SP is confluent, we have: $SP \vdash \sigma(t) = \sigma(r) \iff \sigma(t) \overset{*}{\leftrightarrow} \sigma(r)$. Necessarily, for $t|_{\omega} = g(u_1, \dots, u_n)$ there

is an axiom $\bigwedge_{1 \leq i \leq m} t_i = t'_i \Rightarrow g(v_1, \dots, v_n) \rightarrow v$ in SP and a ground substitution $\rho : V \rightarrow T_\Sigma$ such that $\sigma(t|_\omega) = \rho(g(v_1, \dots, v_n))$, and then, by the confluence of SP , $\sigma(t) \rightarrow_{\mathcal{R}} \sigma(t[\rho(g(v_1, \dots, v_n))]|_\omega) \xrightarrow{*}_{\mathcal{R}} \sigma(r)$ and for all $1 \leq i \leq m$, $\rho(t_i) \xrightarrow{*}_{\mathcal{R}} \rho(t'_i)$, hold. As $Var(t = r) \cap Var(\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v) = \emptyset$, there is a unique ground substitution σ' such that $\sigma'(t) = \sigma(t)$, $\sigma'(r) = \sigma(r)$, $\sigma'(g(v_1, \dots, v_n)) = \rho(g(v_1, \dots, v_n))$, $\sigma'(v) = \rho(v)$ and for all i , $\sigma'(t_i = t'_i) = \rho(t_i = t'_i)$. Hence, $\sigma'(t) = \sigma'(t[g(v_1, \dots, v_n)]|_\omega)$, and then σ' is a unifier of $t|_\omega$ and $g(v_1, \dots, v_n)$. Therefore for all $1 \leq i \leq m$, we have $\sigma'(u_i) \xrightarrow{*}_{\mathcal{R}} \sigma'(v_i)$.

From Theorem 9, we have the expected result as a corollary:

Corollary 10 (Soundness and completeness). *If $\Gamma_0 \vdash_U \Gamma_1 \vdash_U \Gamma_2 \vdash_U \dots$ then for all $i < \omega$, $T_{\Gamma_i, f} = T_0(SP)|_f$.*

By Theorem 9, the LOFT selection procedure is then sound and complete. Moreover, this selection procedure is iterative (see definition 4). However, the selection procedure provided by LOFT is not partitioning because specification axioms are not necessarily disjoint. More precisely, if two rewriting rules (obtained from two different axioms) can be applied simultaneously (at the same position of the same term of the same constraint), then the two resulting subdomains then have some common tests.

Recently, the LOFT selection procedure has been re-used in the GATeL tool [23, 24] that allows to produce test cases from LUSTRE specifications. LUSTRE is a synchronous language widely used in industry to build reactive systems. Hence, the GATeL tool unfolds LUSTRE equations to obtain test subdomains also defined by constraints. Therefore, these constraints are solved such that a test case is randomly built for each subdomain (if not empty).

5 Our selection criteria based on axiom unfolding

In this section, we refine the previous procedure by removing all the constraints such as presenting specifications by reductive and confluent rewrite systems. Here, the only required constraint is that specifications are conditional positive and that's all.

5.1 Unfolding procedure

As in the previous section, the unfolding procedure developed here, has in input:

- an operation $f \in F$,
- a conditional positive specification $SP = (\Sigma, Ax)$ (without any other constraints), and

– a set Γ of Σ -constraint sets.

The first set $\Gamma_0 = \{\{f(x_1, \dots, x_n) = y\}\}$ where $x_i, y \in V$ ($1 \leq i \leq n$).

The unfolding procedure is expressed by the two following inference rules:

Reduce

$$\frac{\Gamma \cup \{\mathcal{C} \cup \{t = t\}\}}{\Gamma \cup \{\mathcal{C}\}}$$

Unfolding

$$\frac{\Gamma \cup \{\mathcal{C} \cup \{\varepsilon\}\}}{\Gamma \cup \bigcup_{c \in Tr(\varepsilon)} \{\mathcal{C} \cup c\}}$$

where $Tr(\varepsilon)$ for $\varepsilon = (t = r)$ a Σ -equation not of the form $u = u$, is the set of Σ -constraint sets defined by:

$$Tr(t = r) = \left\{ \left\{ \begin{array}{l} \left\{ \begin{array}{l} \left\{ u_1 = v_1, \dots, u_n = v_n, t[v]_\omega = r, \alpha_1, \dots, \alpha_m \right\} \\ \left(\begin{array}{l} \sigma(t|_\omega) = \sigma(g(v_1, \dots, v_n)) \text{ (}\sigma \text{ unifier)}, \\ \left(\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v \in Ax \right) \end{array} \right) \\ or \\ \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow v = g(v_1, \dots, v_n) \in Ax \end{array} \right\} \\ \left\{ \begin{array}{l} \left\{ u_1 = v_1, \dots, u_n = v_n \right\}_\omega, t = r[v]_\omega, \alpha_1, \dots, \alpha_m \right\} \\ \left(\begin{array}{l} \sigma(r|_\omega) = \sigma(g(v_1, \dots, v_n)), \text{ (}\sigma \text{ unifier)} \\ \left(\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v \in Ax \right) \end{array} \right) \\ or \\ \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow v = g(v_1, \dots, v_n) \in Ax \end{array} \right\} \end{array} \right\} \cup \left\{ \begin{array}{l} \left\{ \begin{array}{l} \left\{ u_1 = v_1, \dots, u_n = v_n, t[v]_\omega = r, \alpha_1, \dots, \alpha_m \right\} \\ \left(\begin{array}{l} \sigma(t|_\omega) = \sigma(g(v_1, \dots, v_n)) \text{ (}\sigma \text{ unifier)}, \\ \left(\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v \in Ax \right) \end{array} \right) \\ or \\ \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow v = g(v_1, \dots, v_n) \in Ax \end{array} \right\} \\ \left\{ \begin{array}{l} \left\{ u_1 = v_1, \dots, u_n = v_n \right\}_\omega, t = r[v]_\omega, \alpha_1, \dots, \alpha_m \right\} \\ \left(\begin{array}{l} \sigma(r|_\omega) = \sigma(g(v_1, \dots, v_n)), \text{ (}\sigma \text{ unifier)} \\ \left(\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v \in Ax \right) \end{array} \right) \\ or \\ \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow v = g(v_1, \dots, v_n) \in Ax \end{array} \right\} \end{array} \right\} \end{array} \right\}$$

As the definition of $Tr(t = r)$ is based on the subterm relation and unification, this set is computable if the specification SP has a finite set of axioms. Hence, given an equation ε we have the selection criterion C_ε that maps any $T_{\mathcal{C},f}$ to $\{T_{\mathcal{C} \setminus \{\varepsilon\} \cup c}\}_{c \in Tr(\varepsilon)}$ if $\varepsilon \in \mathcal{C}$, $T_{\mathcal{C},f}$ otherwise.

We can observe that the above unfolding procedure is strongly combinatory. This is the result of a complete unfolding on all subterms of both terms t and r . This ensures the completeness of the procedure with respect to the test set $T_0(SP)$ (see the next section). As we saw in section 4, this combinatory can be managed when dealing with very low-level specifications (i.e. executable ones) [6, 22]. The interest here is that the unfolding procedure can be applied to any positive conditional specification with a finite set of axioms. No other

This then means that $SP \vdash \sigma(t) = \sigma(r)$.

- (Completeness) $T_{\Gamma,f} \subseteq T_{\Gamma',f}$. By the definition of the inference rule, this amounts to show

$$T_{\{t=r\},f} \subseteq \bigcup_{c \in Tr(t=r)} T_{c,f}$$

This is equivalent to show that for any ground substitution $\sigma : V \rightarrow T_{\Sigma}$ such that $SP \vdash \sigma(t) = \sigma(r)$, there exists $c \in Tr(t = r)$ such that for all $\varepsilon \in c$, $SP \vdash \sigma(\varepsilon)$.

Note that the unfolding procedure defines a strategy that bounds the search space for proof trees to a given class of trees having a specific structure. Hence, the unfolding procedure defines a proof search strategy which selects proof trees where:

- no instance of transitivity occurs both over instances of symmetry, substitution, and context, and over modus-ponens only when transitivity occurs on the left premise of modus-ponens.
- no instance of modus-ponens occurs over instances of symmetry, substitution, and context.
- no instance of symmetry and context occurs over substitution.

The above inclusion, is proven by showing that there is a proof tree satisfying the above specific structure associated to the statement $SP \vdash \sigma(t) = \sigma(r)$. Actually, we are going to show a stronger result which consists of defining basic proof tree transformations to transform elementary combinations of inference rules, and showing that the global proof tree transformation is terminating. Here, we only give some of these basic transformations of proof trees. The others follow similar transformations.

The case of transitivity over substitution

$$\frac{\frac{\frac{\bigwedge_{i \leq m} \alpha_i \Rightarrow t = u}{\vdots}}{\bigwedge_{i \leq m} \alpha_i \Rightarrow t = v} \quad \frac{\frac{\bigwedge_{i \leq m} \alpha_i \Rightarrow u = v}{\vdots}}{\bigwedge_{i \leq m} \alpha_i \Rightarrow \rho(u) = \rho(v)}}{\bigwedge_{i \leq m} \alpha_i \Rightarrow \rho(t) = \rho(v)} \rightsquigarrow \frac{\frac{\frac{\bigwedge_{i \leq m} \alpha_i \Rightarrow t = u}{\vdots}}{\bigwedge_{i \leq m} \rho(\alpha_i) \Rightarrow \rho(t) = \rho(u)} \quad \frac{\frac{\bigwedge_{i \leq m} \alpha_i \Rightarrow u = v}{\vdots}}{\bigwedge_{i \leq m} \rho(\alpha_i) \Rightarrow \rho(u) = \rho(v)}}{\bigwedge_{i \leq m} \rho(\alpha_i) \Rightarrow \rho(t) = \rho(v)}$$

The case of modus-ponens over substitution

$$\frac{\frac{\frac{\frac{\vdots}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = u}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = u}}{\bigwedge_{2 \leq i \leq m} \rho(\alpha_i) \Rightarrow \rho(t) = \rho(v)} \quad \frac{\vdots}{\alpha_1}}{\frac{\frac{\frac{\frac{\vdots}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = u}}{\bigwedge_{1 \leq i \leq m} \rho(\alpha_i) \Rightarrow \rho(t) = \rho(u)}}{\bigwedge_{2 \leq i \leq m} \rho(\alpha_i) \Rightarrow \rho(t) = \rho(v)} \quad \frac{\vdots}{\rho(\alpha_1)}}}{\bigwedge_{2 \leq i \leq m} \rho(\alpha_i) \Rightarrow \rho(t) = \rho(v)} \rightsquigarrow$$

The case of transitivity over modus-ponens

$$\frac{\frac{\frac{\frac{\vdots}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = u}}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = v}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = v} \quad \frac{\frac{\frac{\vdots}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow u = v}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow u = v}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = v}}{\frac{\vdots}{\alpha_1}} \quad \frac{\frac{\frac{\frac{\vdots}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow t = u}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = u}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = v}}{\frac{\vdots}{\alpha_1}} \quad \frac{\frac{\frac{\frac{\vdots}{\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow u = v}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow u = v}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = v}}{\frac{\vdots}{\alpha_1}}}{\bigwedge_{2 \leq i \leq m} \alpha_i \Rightarrow t = v} \rightsquigarrow$$

Note that basic proof tree transformations are recognized as “distribution” of substitutions instances over other rules, symmetry and context over transitivity and modus-ponens, and modus-ponens over transitivity. Therefore, by using proof terms for proofs, with a recursive path ordering $>^{rpo}$ to order proofs defined from the precedence:

$$\textit{substitution} > \textit{symmetry} \sim \textit{context} > \textit{modus-ponens} > \textit{transitivity}$$

we show that $\rightsquigarrow^* \subseteq >^{rpo}$ and then \rightsquigarrow is terminating^{3, 4}.

Hence, for any statement $SP \vdash \sigma(t) = \sigma(r)$, because $t = r$ is not a tautology (i.e. of the form $u = u$), there is necessarily an axiom $\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v$, a position ω in $\sigma(t)$ or $\sigma(r)$ and a ground substitution ρ such that either $\sigma(t)|_\omega = \rho(g(v_1, \dots, v_n))$ or $\sigma(r)|_\omega = \rho(g(v_1, \dots, v_n))$. As $Var(t = r) \cap Var(\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \dots, v_n) = v) = \emptyset$, there is a unique ground substitution σ' such that $\sigma'(t) = \sigma(t)$, $\sigma'(r) = \sigma(r)$, $\sigma'(g(v_1, \dots, v_n)) = \rho(g(v_1, \dots, v_n))$, $\sigma'(v) = \rho(v)$ and for all i , $\sigma'(\alpha_i) = \rho(\alpha_i)$. Hence, $\sigma'(t) = \sigma'(t[g(v_1, \dots, v_n)]_\omega)$ or $\sigma'(r) = \sigma'(r[g(v_1, \dots, v_n)]_\omega)$, and then σ' is an unifier

³ \rightsquigarrow^* is the transitive and reflexive closure of \rightsquigarrow .

⁴ We refer the interested reader to [1] for a complete proof of $\rightsquigarrow^* \subseteq >^{rpo}$.

of $t|_\omega$ or $r|_\omega$ and $g(v_1, \dots, v_n)$. Therefore, by our global proof tree transformation, there necessarily exists a proof tree associated to the statement $SP \vdash \sigma(t) = \sigma(r)$ with the following form:

$$\begin{array}{c}
 \frac{\frac{\alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow g(v_1, \dots, v_n) = v}{\sigma'(\alpha_1) \wedge \dots \wedge \sigma'(\alpha_m) \Rightarrow \sigma'(g(v_1, \dots, v_n)) = \sigma'(v)} \quad \frac{\vdots}{\sigma'(\alpha_1)} \quad \frac{\vdots}{\sigma'(\alpha_2)} \\
 \frac{\sigma'(\alpha_2) \wedge \dots \wedge \sigma'(\alpha_m) \Rightarrow \sigma'(g(v_1, \dots, v_n)) = \sigma'(v)}{\vdots} \\
 \frac{\sigma'(\alpha_m) \Rightarrow \sigma'(g(v_1, \dots, v_n)) = \sigma'(v)}{\sigma'(g(v_1, \dots, v_n)) = \sigma'(v)} \quad \frac{\vdots}{\sigma'(\alpha_m)} \\
 \frac{\vdots}{\sigma'(t) = \sigma'(t|_\omega)} \quad \frac{\vdots}{\sigma'(t|_\omega) = \sigma'(r)} \\
 \hline
 \sigma(t) = \sigma(r)
 \end{array}$$

Therefore, we have $c = \{u_1 = v_1, \dots, u_n = v_n, t|_\omega = r, \alpha_1, \dots, \alpha_m\}$.

6 Conclusion

Our present work is based on a well-established framework for specification-based testing from algebraic specification [6, 14, 17]. Test case submission is interpreted as the satisfaction by the program of an observable formula. Under some minimal hypotheses, a program can then be considered as correct with respect to the specification if its behaviour matches with at least a model of the specification. The correctness can be ensured by the successful submission of an exhaustive test set, when it exists. In this article, we have been interested in test set selection methods. We have focused on selection criteria for partition testing strategies. It consists in dividing the input domain into subdomains and then in selecting test cases from each of these subdomains. Some relevant properties (soundness, completeness, partition, iterative family) on these selection criteria have been presented. Then we have introduced a general selection procedure based on axioms unfolding and we have shown that for all positive conditional specifications with constructors, this selection procedure is sound and complete. Finally, we have established that this general selection procedure can be restricted when dealing with a subclass of specifications. This restriction remains sound and complete and has been implemented by the LOFT tool [22]. Using this restricted procedure, less subdomains are generated at each unfolding step so the size of test sets becomes easier to manage.

We still have ongoing researches concerning the definition of selection criteria for a larger class of specification including structuration primitives and this work takes inspiration from [20, 21]. Our goal is to be able to propose a framework of functional testing including selection criteria which would be devoted to specification coverage and usable at all steps of the software life cycle, and particularly, at the requirement step.

References

1. M. Aiguier, D. Bahrami, and C. Dubois. On a generalised logicity theorem. In *AISC'2002*, volume 2385 of *L.N.A.I.*, pages 51–64. Springer Verlag, 2002. available at <ftp://ftp.lami.univ-evry.fr/aiguier/>.
2. A. Arnould and P. Le Gall. Test de conformité: une approche algébrique. *Technique et Science Informatiques, Test de logiciel*, vol. 21, n° 9, pages 1219–1242, 2002.
3. A. Arnould, P. Le Gall, and B. Marre. Dynamic testing from bounded data type specifications. In *Dependable Computing - EDCC-2, Second European Dependable Computing Conference*, volume 1150 of *LNCS*, pages 285–302, Taormina, Italy, Octobre 1996. Springer Verlag.
4. G. Bernot. Testing against formal specifications: a theoretical view. In Springer-Verlag LNCS 494, editor, *Proc. TAPSOFT CCPSD*, pages 99–119, July 1990. Brighton.
5. G. Bernot, M.-C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387–405, 1991.
6. Gilles Bernot, Laurent Bouaziz, and Pascale Le Gall. A theory of probabilistic functional testing. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 216–226. ACM Press, 1997.
7. M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2-3):149–186, 1995.
8. Achim D. Brucker and Burkhart Wolff. Symbolic test case generation for primitive recursive functions. In *Formal Approaches to Testing of Software*. 2004. to appear.
9. Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. In *International Conference on Functional Programming*, pages 268–279, 2000.
10. J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *FME'93: Industrial-Strength Formal Methods, First International Symposium of Formal Methods Europe*, volume 670 of *LNCS*, pages 268–284, Odense, Denmark, April 1993. Springer Verlag.
11. M. Doche and V. Wiels. Extended institutions for testing. In *AMAST'2000*, number 1816 in *Lecture Notes in Computer Science*, pages 514–528, 2000.
12. M.C. Gaudel. Testing can be formal, too. In *TAPSOFT'95, International Joint Conference, Theory And Practice of Software Development*, volume 915 of *LNCS*, pages 82–96, Aarhus, Denmark, 1995. Springer Verlag.
13. J.-P. Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In M. Wirsing, editor, *3rd IFIP Conference on Formal Description of Programming Concepts*. Elsevier Science Publishers, 1985.
14. P. Le Gall and A. Arnould. Formal specification and test: correctness and oracle. In *Recent Trends in Data Type Specification*, volume 1130 of *LNCS*, pages 342–358. Springer Verlag, 1996. 11th Workshop on Specification of Abstract Data Types joint with the 9th general COMPASS workshop. Oslo, Norway, September 1995, Selected papers.
15. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

16. Bruno Legeard, Fabien Peureux, and Mark Utting. Controlling test case explosion in test generation from b formal models. *Softw. Test., Verif. Reliab.*, 14(2):81–103, 2004.
17. P. Machado. Testing from structured algebraic specifications. In *AMAST2000*, volume 1816 of *LNCS*, pages 529–544, 2000.
18. Patrícia D. L. Machado and Donald Sannella. Unit testing for casl architectural specifications. In *Mathematical Foundations of Computer Science*, LNCS, pages 506–518. Springer-Verlag, 2002.
19. B. Marre. Toward an automatic test data set selection using algebraic specifications and logic programming. In K. Furukawa, editor, *Eight International Conference on Logic Programming (ICLP'91)*, pages 25–28. MIT Press, 1991.
20. B. Marre and A. Arnould. Test sequences generation from LUSTRE descriptions: GATEL. In *Proceedings of ASE-00: The 15th IEEE Conference on Automated Software Engineering*, pages 229–237, Grenoble, September 2000. IEEE CS Press.
21. B. Marre and B. Blanc. Test selection strategies for lustre descriptions in gatel. In *Proceedings of the Workshop on Model Based Testing (MBT 2004) joint to ETAPS'2004*, volume 111 of *ENTCS*, pages 93–111, 2004. <http://www.sciencedirect.com/science/journal/15710661>.