Specification-Based Testing for CoCasl's Modal Specifications

Delphine Longuet and Marc Aiguier

IBISC CNRS FRE 2873 - University of Évry Val d'Essonne
523 place des terrasses de l'Agora, F-91000 Évry
{delphine.longuet,marc.aiguier}@ibisc.univ-evry.fr

Abstract. Specification-based testing is a particular case of black-box testing, which consists in deriving test cases from an analysis of a formal specification. We present in this paper an extension of the most popular and most efficient selection method widely used in the algebraic framework, called axiom unfolding, to coalgebraic specifications, using the modal logic provided by the CoCasl specification language.

Keywords: Specification-based testing, axiom unfolding, coalgebraic specifications, modal logic, CoCasl.

Black-box testing refers to any method used to validate software systems independently of their implementation. Specification-based testing is a particular case of black-box testing, which consists of the dynamic verification of a system with respect to its specification [1,2,3]. The system under test is executed on a finite subset of its possible input data to check its conformance with respect to the specification requirements.

The testing process is classically divided into two principal phases:

1. The *selection* phase where some selection criteria are defined to split test sets into subsets in order to manage their size.
2. The *generation* phase where some techniques and tools based on constraint solving are defined in order to generate some test cases in each test set to be submitted to the system under test.

In this paper, we are interested in the selection phase. More particularly, we will extend to CoCasl specifications a very popular and very efficient selection method, called *axiom unfolding*, which has extensively been studied in the framework of algebraic specifications [1,2,3,4,5,6,7,8,9].

CoCasl is a coalgebraic extension of the algebraic specification language Casl that allows to specify processes as coalgebraic types dealing with data defined as algebraic types [10]. CoCasl's modal logic is syntactical sugar to express properties on such processes, like safety and fairness properties. We then propose in this paper a selection method for testing dynamic systems specified with CoCasl's modal logic.

The usual approach of black-box testing for dynamic systems is conformance testing [11,12,13,14,15]. In conformance testing, specifications, systems and test

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 356–371, 2007.

purposes are classically represented by input output transition systems. Test cases are then execution traces selected in the specification by using classic techniques from the automata theory such as synchronised product, symbolic evaluation, etc. Recently, some selection methods from test purposes expressed as temporal properties has been investigated (e.g. see [16]). Taking advantage of the fact that specifications are transition systems, model-checking techniques have been used to select trace sets. Here, CoCasl specifications are logical theories. Hence, our selection method, based on axiom unfolding, will be algorithmically defined by defining a search proof strategy. This strategy will enable one to bound the search space for proofs to a given class of trees having a specific structure (see Section 3). However, the aim of the unfolding procedure will not be to find the entire proof of a test purpose φ, but rather to stretch further the execution of the unfolding procedure in order to make increasingly big proof whose remaining lemmas will define a "partition" of φ. Hence, the procedure will be able to be stopped at any time when the obtained partition will be fine enough according to tester's judgement or needs. Completeness of the unfolding procedure will then be established by showing that derivability restricted to the unfolding strategy coincides with the full derivability (i.e. without any specific proof strategy).

The paper is organised as follows. Section 1 briefly presents CoCasl specification language, especially cotype definition. Then CoCasl's modal logic is introduced, and is given a sequent calculus. To set the framework we work within, Section 2 recalls the relevant definitions from [3] we will use in this paper, such as exhaustive test set, and selection criteria and their associated properties. We also prove in this section the important result of the existence of a reference exhaustive test set, allowing to start the selection procedure with. After having recalled in Section 3.1 the general notions of test set and constrained test set from [17], Section 3.2 introduces the unfolding procedure from which is defined a family of selection criteria for CoCasl's modal specifications. Selection criteria thus defined are proved to be sound and complete in Section 3.3.

1 CoCasl's Modal Logic

CoCasl extends Casl specification language by enriching basic specifications with dual forms of algebraic constructs used in Casl to define inductive datatypes. The basic dual form is the cotype construct which is used to specify processes. A cotype declaration defines a coinductive process by declaring *selectors*, also called *observers*, and constructors. Unlike in Casl specifications, constructors here are optional. For example, the two following cotypes can be declared in CoCasl:

spec Moore =
  sorts *In*, *Out*
  cotype *State* ::= (*next* : *In* → *State*;
    observe : *Out*)
end

spec List =
  sort *Nat*
  cotype *List* ::= *empty* |
    insert(*head* :? *Nat*;
      tail :? *List*)
end

The first declaration declares the two observers $next : In \times State \to State$ and $observe : State \to Out$. The second similarly declares observers *head* and *tail* over the cotype *List*, but also constructors $empty : List$ and $insert : Nat \times List \to List$, where *Nat* is an imported sort from the local environment. The parts of the declaration separated by vertical bars are called alternatives. For instance, in the List specification, alternatives are defined by both constructors *empty* and *insert*. Observers may be unary like *observe*, or may have additional parameters, which have to come from the local environment, like *next*. Both observers and constructors may be partial. Observers are partial as soon as the cotype is defined by several alternatives. As cotypes are dual for types, cotype declarations can be strengthened by declaring a cogenerated cotype to restrict the class of models to fully abstract ones, or a cofree cotype to restrict models to the terminal one. For a complete presentation of CoCasl language, the reader may refer to [10].

To express properties on processes declared in CoCasl, a multi-sorted modal logic has been defined in [10], where modalities are defined from observers used to describe system evolutions. All the sorts defined in the cotype are called *non-observable*, while sorts from the local environment are called *observable*. The set of non-observable sorts defines a multi-sorted state space, with observers either directly producing an observable value, or making the system state evolve.

Actually, the modal logic presented here is both a restriction and an extension of the one presented in [10]. This is a restriction because we only consider here quantifier-free formulae. But the logic we present is also an extension because atomic formulae are not restricted to equations but may involve any predicate. The restriction to quantifier-free formulae is due to the fact that existentially quantified formulae are impossible to deal with from a testing point of view. As a matter of fact, testing a formula of the form $\exists x\ \varphi(x)$ requires to exhibit a witness value a such that $\varphi(a)$ is evaluated as "true" by the system under test. Of course, there is no general way to find out such a relevant value, but to simply prove that the system satisfies the property. This led us to conclude that existential properties are not testable [8].

Syntax. A CoCasl *signature* $\Sigma = (S, F, P, V)$ consists of a set S of sorts with a partition S_{obs} and T of observable and non-observable sorts respectively, 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 V of variables. For all operations $f : s_1 \times \ldots \times s_n \to s$ in F and all predicates $p : s_1 \times \ldots \times s_n$ in P, there exists at most one i, $1 \leq i \leq n$, such that $s_i \in T$. We make a distinction between operations coming from the local environment, i.e. operations $f : s_1 \times \ldots \times s_n \to s$ with $s_1, \ldots, s_n, s \in S_{obs}$ on the one hand, and constructors and observers, that are operations $f : s_1 \times \ldots \times s_n \times s \to s'$ with $s \in T$ on the other hand. Constructors have a non-observable result sort, while observers may be with observable result sort $s' \in S_{obs}$ (they are also called attributes) or with non-observable result sort $s' \in T$ (these are also called methods). Constructors and methods are only distinguished from each other thanks to the cotype declaration: the above List declaration declares *empty* and *insert* as constructors, *head* as an observer with observable sort, and *tail* as

an observer with non-observable sort. We call an observer $f : s_1 \times \ldots \times s_n \times s \to s'$ observer of cotype s. The set F of operations names is then a partition $F = F_{obs} \amalg F_\Omega \amalg (F_s)_{s \in T}$ where F_{obs} is the set of operations from the local environment, F_Ω is the set of constructors and for all $s \in T$, F_s is the set of observers of cotype s. Since a cotype may be declared using several alternatives, observers for a given cotype are actually defined for a given alternative of this cotype. For a cotype s having m alternatives, we then have $F_s = \coprod_{1 \leq j \leq m} F_{s,j}$ where $F_{s,j}$ is the set of observers for the j^{th} alternative of cotype s. The set P of predicates is also a partition $P_{obs} \amalg (P_s)_{s \in T}$ where P_{obs} is the set of predicates only involving observable sorts , and for each $s \in T$, P_s is the set of predicates $p : s_1 \times \ldots \times s_n \times s$. The above List declaration gives the following CoCasl signature.

$S_{obs} = \{Nat\} \qquad T = \{List\}$
$F_\Omega = \{empty : List, insert : Nat \times List \to List\} \qquad P_{List} = \{def : List\}$
$F_{List,1} = \emptyset$
$F_{List,2} = \{head : List \to Nat, tail : List \to List\}$

where alternative 1 corresponds to the empty list, and alternative 2 to a list built with constructor *insert*.

Given a signature $\Sigma = (S, F, P, V)$, $T_\Sigma(V)$ is the S-set of *terms with variables* in V defined inductively from variables in V and operations of F: for each operation $f : s_1 \times \ldots \times s_n \to s \in F_{obs} \cup F_\Omega$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_s$, where each $t_i \in T_\Sigma(V)_{s_i}$, $1 \leq i \leq n$; for each observer $f : s_1 \times \ldots \times s_n \times s \to s'$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_{s'}$, where each $t_i \in T_\Sigma(V)_{s_i}$, $1 \leq i \leq n$. Notice that, for observers, the sort s has been removed. This allows to consider states as implicit, as usual with modal logic. The set of *ground terms* T_Σ is defined as the set of terms built over the empty set of variables $T_\Sigma(\emptyset)$. A *substitution* is any mapping $\sigma : V \to T_\Sigma(V)$ that preserves sorts. Substitutions are naturally extended to terms with variables and then to formulae.

Σ-atomic formulae are sentences of the form $p(t_1, \ldots, t_n)$ where $p : s_1 \times \ldots \times s_n \in P_{obs}$ or $p : s_1 \times \ldots \times s_n \times s \in P_s$, and $t_i \in T_\Sigma(V)_{s_i}$ for each i, $1 \leq i \leq n$. A term t with non-observable sort leads to modalities $[t]$, $\langle t \rangle$, $[t*]$ and $\langle t* \rangle$, intuitively meaning "all next state", "some next state", "always" and "eventually", respectively. Modalities can be extended to finite sequences $\{t_1, \ldots, t_n\}$, where $[\{t_1, \ldots, t_n\}]\varphi$ and $\langle\{t_1, \ldots, t_n\}\rangle\varphi$ stand respectively for the conjunction and the disjunction of the modal formulae obtained for the corresponding individual modalities. *Formulae* are then built following the syntax:

$$\varphi, \psi ::= true \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid [t]\varphi \mid \langle t \rangle\varphi \mid [t*]\varphi \mid \langle t* \rangle\varphi$$
$$\mid [\{t_1, \ldots, t_n\}]\varphi \mid \langle\{t_1, \ldots, t_n\}\rangle\varphi \mid [\{t_1, \ldots, t_n\}*]\varphi \mid \langle\{t_1, \ldots, t_n\}*\rangle\varphi$$

The set of modalities is denoted by $M_\Sigma(V)$. $For(\Sigma)$ is the set of all Σ-formulae. A *specification* $Sp = (\Sigma, Ax)$ consists of a signature Σ and a set Ax of formulae often called *axioms*. The List declaration above generates, besides the signature we gave, the following axioms, as well as the five axioms specifying that the equality predicate is the existential equality:

$$\neg def(head(empty)) \qquad head(insert(n,l)) = n$$
$$\neg def(tail(empty)) \qquad tail(insert(n,l)) = l$$
$$x = x \qquad t = t' \Rightarrow t' = t \qquad t = t' \wedge t' = t'' \Rightarrow t = t''$$
$$t_1 = t'_1 \wedge \cdots \wedge t_n = t'_n \wedge def(f(t_1,\ldots,t_n)) \Rightarrow f(t_1,\ldots,t_n) = f(t'_1,\ldots,t'_n)$$
$$def(f(t_1,\ldots,t_n)) \Rightarrow def(t_i) \quad \text{(strictness)} \qquad t = t' \Rightarrow def(t) \quad \text{(definability)}$$

Semantics. Given a signature $\Sigma = (S, F, P, V)$, we denote by Σ_{obs} the "observable subsignature" $(S, F_{obs} \amalg F_\Omega, P_{obs}, V)$ of Σ. A Σ_{obs}-model $\mathcal{A}$ is then a first-order structure, that is an S-indexed set A, equipped for each operation name $f : s_1 \times \ldots \times s_n \to s \in F_{obs} \amalg F_\Omega$ with a mapping $f^{\mathcal{A}} : A_{s_1} \times \ldots \times A_{s_n} \to A_s$, and for each predicate name $p : s_1 \times \ldots \times s_n \in P_{obs}$ with an n-ary relation $p^{\mathcal{A}} \subseteq A_{s_1} \times \ldots \times A_{s_n}$.

Since several cotypes can be declared in CoCasl, the set of states E is said multi-sorted and is defined as a product $E = \prod_{s \in T} E_s$ where for each $s \in T$, $E_s = A_s$. Σ-models are then coalgebras $(E, \alpha : E \to \mathcal{F}E)$ of the functor $\mathcal{F}$ such that $\mathcal{F}E = \prod_{s \in T} \mathcal{F}E_s$ and which, for each $s \in T$, associates to E_s the set $\mathcal{F}E_s$ defined as follows:

$$\mathcal{F}E_s = \coprod_{1 \le j \le m} \left(\prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j} \\ s' \in S_{obs}}} A_{s'}^{A_{s_1} \times \ldots \times A_{s_n}} \times \prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j} \\ s' \in T}} E_{s'}^{A_{s_1} \times \ldots \times A_{s_n}} \right) \times \prod_{p : s_1 \times \ldots \times s_n \times s \in P_s} 2^{A_{s_1} \times \ldots \times A_{s_n}}$$

where sort s is defined by m alternatives. [1] We denote by $Mod(\Sigma)$ the category whose objects are Σ-models, i.e. the category $\mathsf{Coalg}(\mathcal{F})$ of coalgebras over $\mathcal{F}$.

Given a Σ-model (E, α) over a first-order structure $\mathcal{A}$, we denote by $\_^A : T_{\Sigma_{obs}} \to A$ the unique homomorphism that maps any Σ_{obs} ground term $f(t_1, \ldots, t_n)$ to its value $f^{\mathcal{A}}(t_1^A, \ldots, t_n^A)$. A Σ-model is said *reachable on data* if $\_^A$ is surjective.

Given a Σ-model (E, α), a Σ-interpretation in A is any mapping $\nu : V \to A$ preserving sorts. Given an interpretation of variables ν and a state $e = (e_s)_{s \in T} \in E$, the interpretation of terms in $T_\Sigma(V)$ $\nu_e^\natural : T_\Sigma(V) \to M$ is built in the usual way for variables and operations in $F_{obs} \cup F_\Omega$, and in the following way for observers: if $f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j}$ is an observer with observable result sort then $\nu_e^\natural(f(t_1, \ldots, t_n)) = (\pi_f \circ \kappa_j \circ \pi_s \circ \alpha)(e)(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$, where: $\pi_s : E \to E_s$ is the canonical projection to the s-sorted part of a state; assuming that the sort s is declared by j alternatives, κ_j is the canonical injection to alternative j; and π_f is the canonical projection from alternative j of E_s to the interpretation of f; if $f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j}$ is an observer with non-observable result sort, then $\nu_e^\natural(f(t_1, \ldots, t_n)) = e'$ such that $e' = (e'_s)_{s \in T} \in E$ with $e'_{s''} = e_{s''}$ for all $s'' \neq s'$, and $e_{s'} = (\pi_f \circ \kappa_j \circ \pi_s \circ \alpha)(e)(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$. By abuse of notation, the extension $\nu_e^\natural$ of ν will be denoted by ν_e.

The *satisfaction* of a Σ-formula φ by (E, α) for an interpretation ν and a state e, denoted by $(E, \alpha) \models_{\nu,e} \varphi$, is inductively defined on the structure of φ: $(E, \alpha) \models_{\nu,e} true$ always holds; $(E, \alpha) \models_{\nu,e} p(t_1, \ldots, t_n)$ for $p \in P_{obs}$ if and

[1] If A and B are two sets, we denote by B^A the set of all mappings from A to B.

only if $(\nu_e(t_1),\ldots,\nu_e(t_n)) \in p^{\mathcal{A}}$; $(E,\alpha) \models_{\nu,e} p(t_1,\ldots,t_n)$ for $p \in P_s$ if and only if $(\nu_e(t_1),\ldots,\nu_e(t_n)) \in \pi_p \circ \pi_s(e)$; $(E,\alpha) \models_{\nu,e} [t]\psi$ if and only if for all $e' \in E$ such that $\nu_e(t) = e'$, $(E,\alpha) \models_{\nu,e'} \psi$. The other modalities can be defined as derived notions. Actually, we have the following elementary equivalences:[2] $\langle t\rangle\varphi \equiv \neg[t]\neg\varphi$; $[t*]\varphi \equiv \varphi \wedge [t][t*]\varphi$; $[\{t_1,\ldots,t_n\}]\varphi \equiv [t_1]\varphi \wedge \ldots \wedge [t_n]\varphi$. Boolean connectives are interpreted as usual. (E,α) *validates* a formula φ, denoted by $(E,\alpha) \models \varphi$, if and only if for every interpretation $\nu : V \to A$ and every state $e \in E$, $(E,\alpha) \models_{\nu,e} \varphi$. Given $\Psi \subseteq For(\Sigma)$ and two Σ-models (E,α) and (E',α'), (E,α) is *Ψ-equivalent to* (E',α'), denoted by $(E,\alpha) \equiv_\Psi (E',\alpha')$, if and only if we have: $\forall \varphi \in \Psi,\ (E,\alpha) \models \varphi \Leftrightarrow (E',\alpha') \models \varphi$. Given a specification $Sp = (\Sigma, Ax)$, a Σ-model (E,α) is an *Sp-model* if for every $\varphi \in Ax$, $(E,\alpha) \models \varphi$. $Mod(Sp)$ is the full subcategory of $Mod(\Sigma)$, objects of which are all Sp-models. A Σ-formula φ is a *semantic consequence* of a specification $Sp = (\Sigma, Ax)$, denoted by $Sp \models \varphi$, if and only if for every Sp-model (E,α), we have $(E,\alpha) \models \varphi$. $Sp^\bullet$ is the set of all semantic consequences.

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

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

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

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

$$\frac{\Gamma \mathrel{|\!\sim} \varphi}{[t]\Gamma \mathrel{|\!\sim} [t]\varphi}\,\mathsf{Nec} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta}{\sigma(\Gamma) \mathrel{|\!\sim} \sigma(\Delta)}\,\mathsf{Subs} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta,\varphi \quad \Gamma',\varphi \mathrel{|\!\sim} \Delta'}{\Gamma,\Gamma' \mathrel{|\!\sim} \Delta,\Delta'}\,\mathsf{Cut}$$

where $[t]\Gamma = \{[t]\gamma \mid \gamma \in \Gamma\}$, $\langle t\rangle\Gamma = \{\langle t\rangle\gamma \mid \gamma \in \Gamma\}$ and $\sigma(\Gamma) = \{\sigma(\gamma) \mid \gamma \in \Gamma\}$. This calculus is the standard Gentzen sequent calculus for modal logic **K** which underlies CoCasl's logic, from which we removed the axiom scheme called Kripke distribution axiom: $[t](\varphi \Rightarrow \psi) \Rightarrow ([t]\varphi \Rightarrow [t]\psi)$, since it is of no interest for our unfolding procedure. From rule Nec, we can derive the following rules, which will be helpful later:

$$\frac{\Gamma \mathrel{|\!\sim} \varphi}{[t*]\Gamma \mathrel{|\!\sim} [t*]\varphi}\,\mathsf{Nec}^* \qquad \frac{\Gamma \mathrel{|\!\sim} \varphi}{[\{t_1,\ldots,t_n\}]\Gamma \mathrel{|\!\sim} [\{t_1,\ldots,t_n\}]\varphi}\,\mathsf{Nec}^n$$

$$\frac{\Gamma \mathrel{|\!\sim} \varphi,\Delta}{[t]\Gamma \mathrel{|\!\sim} [t]\varphi,\langle t\rangle\Delta} \qquad \frac{\Gamma,\varphi \mathrel{|\!\sim} \Delta}{[t]\Gamma,\langle t\rangle\varphi \mathrel{|\!\sim} \langle t\rangle\Delta} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta}{[t]\Gamma \mathrel{|\!\sim} \langle t\rangle\Delta}$$

In order to manipulate less complex formulae, we take advantage of the fact that the inference rules associated to Boolean connectives define an automatic process that allows to transform any sequent $\mathrel{|\!\sim} \varphi$, where φ is a modal formula, into a set of sequents $\Gamma \mathrel{|\!\sim} \Delta$ where every formula in Γ and Δ is of the form

[2] Two formulae φ and ψ are said elementarily equivalent, denoted by $\varphi \equiv \psi$, if and only if for each Σ-model (E,α), for each interpretation ν and every state e, $(E,\alpha) \models_{\nu,e} \varphi \Leftrightarrow (E,\alpha) \models_{\nu,e} \psi$.

$\alpha_1 \dots \alpha_n \psi$, where $\alpha_i \in M_\Sigma(V)$ for all i, $1 \leq i \leq n$, and $\psi \in For(\Sigma)$ is a formula not beginning with a modality. Let us call such sequents *normalised sequents*.

More precisely, these normalised sequents are obtained by eliminating every boolean connectives which is not in the scope of a modal operator with the help of the above sequent calculus. Such a syntactic transformation can be done since the inference rules associated to boolean connectives are reversible: given an inference rule $\frac{\varphi_1 \dots \varphi_n}{\varphi}$ amongst {Left-@, Right-@} where $@ \in \{\neg, \wedge, \vee, \Rightarrow\}$, we have $\bigwedge_{1 \leq i \leq n} \varphi_i \equiv \varphi$. Then, applying reversed inference rules for boolean connectives to any sequent leads to an equivalent set of normalised sequents, which allows to only deal with normalised sequents. Therefore, in the following, we will suppose that specification axioms are normalised sequents. These transformations enable us to remove the rules associated to boolean connectives from the unfolding procedure.

Example 1 (Running Example)
To illustrate our approach, we continue here the specification of the LIST cotype. We specify two additional observers $odd : List \to List$ and $even : List \to List$ which give a list containing all the elements occurring in oddly numbered places of the original list, in evenly numbered places respectively. We have the following modal axioms:[3]

- $head = n \Leftrightarrow \langle odd \rangle head = n$
- $[odd][tail]\varphi \Leftrightarrow [tail][tail][odd]\varphi$
- $[even]\varphi \Leftrightarrow [tail][odd]\varphi$

We don't specify the data part here, since we are only interested in specifying the process part. Axioms are then transformed into normalised sequents, as explained above. For example, the first axiom $head = n \Leftrightarrow \langle odd \rangle head = n$, which is equivalent to the formula $head = n \Rightarrow \langle odd \rangle head = n \wedge \langle odd \rangle head = n \Rightarrow head = n$, leads to the two sequents $head = n \vdash \langle odd \rangle head = n$ and $\langle odd \rangle head = n \vdash head = n$.

1. $head = n \vdash \langle odd \rangle head = n$
2. $\langle odd \rangle head = n \vdash head = n$
3. $[odd][tail]\varphi \vdash [tail][tail][odd]\varphi$
4. $[tail][tail][odd]\varphi \vdash [odd][tail]\varphi$
5. $[even]\varphi \vdash [tail][odd]\varphi$
6. $[tail][odd]\varphi \vdash [even]\varphi$

Due to lack of space, we don't give a more complex and larger example here, but another example dealing with the CoCASL's modal specification of a cash machine may be found in the long version of this paper [18].

2 Testing from Logical Specifications

The work presented in Section 3 comes within the general framework of testing from formal specifications defined in [3]. So that the paper is as self-contained as possible, we succinctly introduce this framework and we instantiate it to the CoCASL's formalism presented in Section 1.

[3] The second and third axioms actually are axiom schemes, i.e. they denote the sets of all their instances with any formula substituted for φ.

Following previous works [1,3,7,9,19], given a specification $Sp = (\Sigma, Ax)$, the basic assumption is that the system under test can be assimilated to a model of the signature Σ. Test cases are then Σ-formulae which are semantic consequences of the specification Sp (i.e. elements of $Sp^\bullet$). As these formulae are to be submitted to the system, test case interpretation is defined in terms of formula satisfaction. When a test case is submitted to a system, it has to yield a verdict (success or failure). Hence, test cases have to be directly interpreted as "true" or "false" by a computation of the system. Obviously, systems can't deal with formulae containing non-instantiated variables, so test cases have to be ground formulae, that is formulae where all variables have been replaced with actual values. These "executable" formulae are called *observable*. Then a test case is any observable semantic consequence. If we denote by $Obs \subseteq For(\Sigma)$ the set of observable formulae, then a *test set* T is any subset of $Sp^\bullet \cap Obs$. Since the system under test is considered to be a Σ-model P, T is said to be *successful* for P if and only if $\forall \varphi \in T, P \models \varphi$.

The interpretation of test cases submission as a success or failure is related to the notion of system correctness. Following an observational approach [20], to be qualified as correct with respect to a specification Sp, a system is required to be observationally equivalent to a model of $Mod(Sp)$ up to the observable formulae of Obs, that is, they have to validate exactly the same observable formulae.

Definition 1 (Correctness). *P is* correct *for Sp via Obs, denoted by $Correct_{Obs}(P, Sp)$, if and only if there exists a model $\mathcal{M}$ in $Mod(Sp)$ such that $\mathcal{M} \equiv_{Obs} P$.*[4]

A test set allowing to establish the system correctness is said *exhaustive*. Formally, an exhaustive set is defined as follows:

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

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

The existence of an exhaustive test set means that systems belonging to the class $\mathcal{K}$ are testable with respect to Sp via Obs, since correctness can be asymptotically approached by submitting a (possibly infinite) test set. Hence, an exhaustive test set is appropriate to start the process of selecting test sets. However, such an exhaustive set does not necessarily exist, depending on the nature of both specifications and systems (whence the usefulness of subclass $\mathcal{K}$ of systems in Definition 2), and on the chosen set of observable formulae. For instance, we will need here to assume that the system under test is reachable on data. Among all the test sets, the biggest one is the set $Sp^\bullet \cap Obs$ of observable semantic consequences of the specification. Hence, to start the selection phase of the testing process, we first have to show that $Sp^\bullet \cap Obs$ is exhaustive. This holds for every system reachable on data as stated by Theorem 1.

[4] Equivalence of Σ-models with respect to a set of formulae is defined in Section 1.

Theorem 1. *Let $Sp = (\Sigma, Ax)$ be a specification. Then the test set $Sp^\bullet \cap Obs$ is exhaustive for every model reachable on data.*

Idea of the proof. Considering a system $\mathcal{S}$ reachable on data, we use classic results of the coalgebra theory [21] to build a final coalgebra elementary equivalent to $\mathcal{S}$ with respect to Obs, and then show that a well-chosen subcoalgebra of it (also elementary equivalent to $\mathcal{S}$ up to Obs) is a model of specification Sp.

The entire proof may be found in [18]. □

The challenge, when dealing with specifications defined as logical theories, consists in managing the size of $Sp^\bullet \cap Obs$, which is most of the time infinite. In practice, experts apply some selection criteria in order to extract a set of test cases of sufficiently reasonable size to be submitted to the system. The underlying idea is that all test cases satisfying a considered selection criterion reveal the same class of incorrect systems, intuitively those corresponding to the fault model captured by the criterion. For example, the criterion called *uniformity hypothesis* states that test cases in a test set all have the same power to make the system fail.

A classic way to select test data with a selection criterion C consists in splitting a given starting test set T into a family of test subsets $\{T_i\}_{i \in I_{C(T)}}$ such that $T = \cup_{i \in I_{C(T)}} T_i$ holds. A test set satisfying such a selection criterion simply contains at least one test case for each non-empty subset T_i. 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 the method that we will use in this paper to select test data, known under the term of *partition testing*.

Definition 3 (Selection criterion). *A* selection criterion *C is a mapping $\mathcal{P}(Sp^\bullet \cap Obs) \to \mathcal{P}(\mathcal{P}(Sp^\bullet \cap Obs))$.*[5] *For a test set T, we note $|C(T)| = \cup_{i \in I_{C(T)}} T_i$ where $C(T) = \{T_i\}_{i \in I_{C(T)}}$. T'* satisfies *C* applied to *T, noted by $T' \sqsubset C(T)$ if and only if: $\forall i \in I_{C(T)}, T_i \neq \emptyset \Rightarrow T' \cap T_i \neq \emptyset$.*

To be pertinent, a selection criterion should ensure some properties between the starting test set and the resulting family of test sets:

Definition 4 (Properties). *Let C be a selection criterion and T be a test set. C is said* sound for *T if and only if $|C(T)| \subseteq T$. C is said* complete for *T if and only if $|C(T)| \supseteq T$.*

These properties 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 no test from the initial test set is lost. A sound and complete selection criterion then preserves exactly all the test cases of the initial test set, up to the notion of equivalent test cases.

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

3 Selection Criteria Based on Axiom Unfolding

In this section, we study the problem of test case selection for quantifier-free modal CoCasl specifications, by adapting a selection criteria based on unfolding of quantifier-free first-order formulae recently defined in the first-order specifications setting [17].

3.1 Test Sets for Modal CoCasl Formulae

We recall here general definitions of test sets from [17]. The selection method that we are going to define takes inspiration from classic methods that split the initial test set of any formula considered as a test purpose.

Succinctly, for a modal CoCasl formula φ, our method consists in splitting the initial test set for φ into many test subsets, called *constrained test sets for* φ, and choosing any input in each non-empty subset. First, let us define what test set and constrained test set for a modal CoCasl formula are.

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

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

Note that φ may be any formula, not necessarily in $Sp^\bullet$. When $\varphi \notin Sp^\bullet$ then $T_\varphi = \emptyset$. Constrained test sets will be sets generated by our unfolding procedure. They are defined as follows.

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

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

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

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

3.2 Unfolding Procedure

Given a test purpose φ, the unfolding procedure will then replace the initial constrained test purpose $\langle(\{\varphi\}, Id),\varphi\rangle$ with a set of constrained test purposes $\langle(\mathcal{C},\sigma),\varphi\rangle$. This will be achieved by matching (up to unification), formulae in $\mathcal{C}$ for any constrained test purpose $\langle(\mathcal{C},\sigma),\varphi\rangle$ with the specification axioms. Hence, step by step, we will see that the unfolding procedure is building a proof tree of conclusion φ having the following structure :

- no instance of cut occurs over instances of substitution and necessitation
- no instance of substitution occurs over instances of necessitation

– there is no instance of cut with two instances of cut occurring over it.

Hence, the unfolding procedure will only involve cut, substitution and necessitation rules. In order to allow many applications of the necessitation rule at each step of the unfolding procedure, let us define the following relation $\mathcal{R}$ over tuples of modality sequences.

Definition 7. *Let $p, q \in \mathbb{N}$. $\mathcal{R} \subseteq (M_\Sigma(V)^*)^p \times (M_\Sigma(V)^*)^q$ is defined for all $(M_1, \ldots, M_p) \in (M_\Sigma(V)^*)^p$ and $(N_1, \ldots, N_q) \in (M_\Sigma(V)^*)^q$ as follows:*
$(M_1, \ldots, M_p)\mathcal{R}(N_1, \ldots, N_q)$ if and only if

1. *there exists $n \in \mathbb{N}$ such that for all i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, there exists $\alpha_1^i, \ldots, \alpha_n^i$ and $\beta_1^i, \ldots, \beta_n^i$ such that $M_i = \alpha_1^i \ldots \alpha_n^i$ and $N_j = \beta_1^j \ldots \beta_n^j$*
2. *for all l, $1 \leq l \leq n$, $\alpha_l^1, \ldots, \alpha_l^p$ and $\beta_l^1, \ldots, \beta_l^q$ are such that:*
   (a) *there exists $t \in T_\Sigma(V)$ such that for all i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, α_l^i and β_l^j all equal to $[t]$ or $\langle t \rangle$, or α_l^i and β_l^j all equal to $[t*]$ or $\langle t* \rangle$*
   (b) *for all i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, $\alpha_l^i = [t]$ and $\beta_l^j = \langle t \rangle$ (resp. $\alpha_l^i = [t*]$ and $\beta_l^j = \langle t* \rangle$), except perhaps either for one k, $1 \leq k \leq p$, such that $\alpha_l^k = \langle t \rangle$ (resp. $\alpha_l^k = \langle t* \rangle$), or for one k, $1 \leq k \leq q$, such that $\beta_l^k = [t]$ (resp. $\beta_l^k = [t*]$).*

This relation then ensures the following proposition.

Proposition 1. *Let $\gamma_1, \ldots, \gamma_p \mathrel{|\!\sim} \delta_1, \ldots, \delta_q$ be any sequent. Let $(M_1, \ldots, M_p) \in (M_\Sigma(V)^*)^p$ and $(N_1, \ldots, N_q) \in (M_\Sigma(V)^*)^q$ such that $(M_1, \ldots, M_p)\mathcal{R}(N_1, \ldots, N_q)$. Then there exists a proof tree of conclusion $M_1\gamma_1, \ldots, M_p\gamma_p \mathrel{|\!\sim} N_1\delta_1, \ldots, N_q\delta_q$ composed only of instances of the necessitation rule.*

We can now proceed with the presentation of the unfolding procedure. The procedure inputs are:

– a modal CoCasl specification $Sp = (\Sigma, Ax)$ where axioms of Ax have been transformed into normalised sequents (see Section 1)
– a modal formula φ representing the test purpose $\langle(\{\varphi\}, Id), \varphi\rangle$
– a family Ψ of couples $(\mathcal{C}, \sigma)$ where $\mathcal{C}$ is a set of Σ-constraints in the form of normalised sequents, and σ is a substitution $V \to T_\Sigma(V)$.

Test sets for φ with respect to couples $(\mathcal{C}, \sigma)$ are naturally extended to Ψ as follows: $T_{\Psi,\varphi} = \bigcup_{(\mathcal{C},\sigma)\in\Psi} T_{(\mathcal{C},\sigma),\varphi}$. The first set Ψ_0 only contains the couple composed of the set of normalised sequents obtained from the modal formula φ under test and the identity substitution.

The unfolding procedure is expressed by the two following rules:[6]

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

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

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

where $Tr(\phi)$ for $\phi = \gamma_1, \ldots, \gamma_m \mathrel{|\!\sim} \delta_1, \ldots, \delta_n$ is the set of couples

$$\{(c, \sigma) \mid Cond(c, \sigma)\}$$

where, for each couple, c is the following set

$$\{\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m), \sigma(N'_i\zeta_i) \mathrel{|\!\sim} \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1\le i\le k}$$
$$\bigcup$$
$$\{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m) \mathrel{|\!\sim} \sigma(M'_i\xi_i), \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1\le i\le l}$$

and the condition on (c, σ) denoted by $Cond(c, \sigma)$ is the following: there exists an axiom $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mathrel{|\!\sim} \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q \in Ax$ with $k, l \in \mathbb{N}$, $1 \le p \le m$ and $1 \le q \le n$, and there exists a unifier σ such that

- for all $1 \le i \le p$, there exists $M_i \in M_\Sigma(V)^*$ such that $\sigma(M_i\psi_i) = \sigma(\gamma_i)$,
- for all $1 \le i \le q$, there exists $N_i \in M_\Sigma(V)^*$ such that $\sigma(N_i\varphi_i) = \sigma(\delta_i)$,
- for all $1 \le i \le l$ and for all $1 \le j \le k$, $M'_i, N'_j \in M_\Sigma(V)^*$ with $(M_1, \ldots, M_p, M'_1, \ldots, M'_l)\mathcal{R}(N_1, \ldots, N_q, N'_1, \ldots, N'_k)$

The **Reduce** rule eliminates tautologies from constraints sets (up to substitution), which are without interest for the unfolding procedure. The **Unfold** rule is closely related to the one given in [17] although more complicated because of modalities. Roughly speaking, this rule consists in replacing the formula ϕ with the set c of constraints, ϕ being the conclusion of an instance of the Cut rule, and the constraints in c being the premisses of this rule instance which do not directly come from a substitution (up to some applications of the necessitation rule) of an axiom of the specification. The relevance of the method is due to the fact that testing $\sigma(\phi)$ comes to test the formulae of c, which will be proved in the next subsection. The particular case where no formula has to be cut is taken into account, since k and l may be equal to zero. $Tr(\phi)$ is then a couple $(\emptyset, \sigma)$, and it is the last step of unfolding for this formula.

Each unification with an axiom leads to as much couples (c, σ) as there are ways to instantiate $M'_1, \ldots, M'_l$ and $N'_1, \ldots, N'_k$ so that $(M_1, \ldots, M_p, M'_1, \ldots, M'_l)$ and $(N_1, \ldots, N_q, N'_1, \ldots, N'_k)$ belong to $\mathcal{R}$. So the initial formula ϕ is replaced with, at least, as much sets of formulae as there are axioms to which it can be unified. The definition of $Tr(\phi)$ being based on unification, this set is computable if the specification Sp has a finite set of axioms. Therefore, given an atomic formula ψ, we have the selection criterion C_ψ that maps any $T_{(\mathcal{C},\sigma'),\varphi}$ to $(T_{(\sigma(\mathcal{C}\setminus\{\phi\})\cup c,\sigma\circ\sigma'),\varphi})_{(c,\sigma)\in Tr(\phi)}$ if $\phi \in \mathcal{C}$, and to $T_{\mathcal{C},\varphi}$ otherwise.

We write $\langle \Psi, \varphi\rangle \vdash_U \langle \Psi', \varphi\rangle$ to mean that Ψ' can be derived from Ψ by applying **Reduce** or **Unfold**. An unfolding procedure is then any program, whose inputs are a CoCasl's modal specification Sp and a modal formula φ, and uses the above inference rules to generate the sequence $\langle \Psi_0, \varphi\rangle \vdash_U \langle \Psi_1, \varphi\rangle \vdash_U \langle \Psi_2, \varphi\rangle \ldots$

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 in the introduction, 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 are not true, and then the associated test set is empty.

Example 2 (Lists). Let us suppose that we want to test the formula $[even][tail]$ $head = a \Rightarrow [tail][tail][even]head = b$. Then, to perform the first step of the unfolding procedure on the initial family of couples:

$$\Psi_0 = \{(\{[even][tail]head = a \mathrel{|\!\sim} [tail][tail][even]head = b\}, Id)\}$$

leads to the following family of couples:

$$\begin{aligned}
\Psi_1 = \{&(\{[even][tail]\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), && (1)\\
&(\{\langle even\rangle[tail]\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), && (1)\\
&(\{[even]\langle tail\rangle\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), && (1)\\
&(\{\langle even\rangle\langle tail\rangle\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), && (1)\\
&(\{[tail][odd][tail]head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), && (5)\\
&(\{[even][tail]head = n_0 \mathrel{|\!\sim} [tail][tail][even]\langle odd\rangle head = m_0\}, \sigma_2), && (2)\\
&(\{[even][tail]head = n_0 \mathrel{|\!\sim} [tail][tail][tail][odd]head = m_0\}, \sigma_2)\} && (6)
\end{aligned}$$

where $\sigma_1 : a \mapsto n_0, b \mapsto m_0, n \mapsto n_0$ and $\sigma_2 : a \mapsto n_0, b \mapsto m_0, n \mapsto m_0$. Each couple of Ψ_1 is labelled by the number of the axiom used for the unfolding of the initial formula.

The first four couples of Ψ_1 come from the unification of the initial formula with axiom (1). Since $\sigma_1(M_1\psi_1) = \sigma_1(\gamma_1)$, where $M_1 = [even][tail]$, ψ_1 is the formula $head = n$ and γ_1 is $head = a$, the resulting constraints are the sequents $\sigma_1(N_1'\zeta_1) \mathrel{|\!\sim} \sigma_1(\delta_1)$ where ζ_1 is the formula $\langle odd\rangle head = n$, δ_1 is $[tail][tail][even]head = b$, and N_1' must be such that $M_1 \mathcal{R} N_1$. According to the definition of $\mathcal{R}$, several N_1 suit, namely $[even][tail]$, $\langle even\rangle[tail]$, $[even]\langle tail\rangle$ and $\langle even\rangle\langle tail\rangle$, whence the four constraints generated by the unification with axiom (1).

Notice that the formula under test is a consequence of the specification if and only if $a = b$. The unfolding may then generate two kinds of constrained test sets: those whose substitution σ is such that $\sigma(a) = \sigma(b)$, which will lead to test cases since they are consequences of the specification, and those where $\sigma(a) \neq \sigma(b)$, which are not test cases. Here, when a constraint is unified with both sides of axiom (1) or (2), the substitution collapses a and b and the resulting constrained test set is a potential test case.

The unfolding procedure can not distinguish between these two kinds of constrained test sets, but this distinction will be done before submitting them to the

system, by applying a ground substitution ρ to any formula in constrained test purposes. Since, by definition, $\rho(\psi)$ has to be a consequence of the specification, constrained test sets where $\sigma(a) \neq \sigma(b)$ will not be submitted to the system.

The application of the procedure on another example may be found in [18].

Until now, the unfolding procedure has been defined in order to cover the 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 modal formulae with respect to their size, as follows:

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

$$\Phi_{n+1} = \{ \mid\sim \neg\psi,\ \mid\sim [m]\psi,\ \mid\sim \psi_1 @ \psi_2 \mid m \in M_\Sigma(V), @ \in \{\wedge, \vee, \Rightarrow\}, \psi, \psi_1, \psi_2 \in \Phi_n\}$$

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.

3.3 Soundness and Completeness

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.

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

Idea of the proof. To prove the soundness of the procedure comes to prove that the initial formula φ can be derived from the constraints replacing it in the procedure. Thus we prove that the test set obtained by the application of the procedure does not add new test cases. Then, to prove the completeness of the procedure, we prove that there necessarily exists a proof tree of conclusion φ having a certain structure, and then that the procedure generates all possible constraints for testing φ. We thus prove that no test cases are lost. As explained just before, 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 occurs over instances of substitution and necessitation
- no instance of substitution occurs over instances of necessitation
- there is no instance of cut with two instances of cut occurring over it.

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.

The entire proof may be found in [18]. □

4 Conclusion

In this paper, we have extended the method for selecting test cases known as axiom unfolding to coalgebraic specifications of dynamic systems. As in the algebraic specifications setting, our unfolding procedure consists in dividing the initial test set for a formula into subsets. The generation of a test set for this formula then arises from the selection of one test case in each resulting subset. We have proved this procedure to be sound and complete, so that test cases are preserved at each step. We have also proved the exhaustiveness of the set of observable consequences of the specification for every reachable system, and proposed a strategy to cover this exhaustive test set.

Ongoing research concerns several aspects. First, we have to specialize our unfolding procedure by handling (strong and existential) equality in a more efficient way. We lose here the strong equality, and the advantage of equality being a congruence. Then we have to extend this work to the very recent extension of CoCasl logic [22]. This logic deals with modalities at a more abstract level than the one presented here, using Pattinson's predicate liftings. This extension of CoCasl allows to specify in several modal logics that were not handled with basic CoCasl, such as probabilistic modal logic. Defining testing for such an extension of CoCasl would allow us to handle a larger variety of modal formalisms in our framework. Another important future work will be to include structuration, such as provided by Casl and CoCasl languages, in our framework, both on its first-order side, by extending our work developed in [17], and on its coalgebraic side, by extending the present work. This work will surely take inspiration from [6,23].

References

1. Bernot, G., Gaudel, M.C., Marre, B.: Software testing based on formal specifications: a theory and a tool. Software Engineering Journal 6(6), 387–405 (1991)
2. Gaudel, M.C.: Testing can be formal, too. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 82–96. Springer, Heidelberg (1995)
3. Le Gall, P., Arnould, A.: Formal specification and test: correctness and oracle. In: Haveraaen, M., Dahl, O.-J., Owe, O. (eds.) Recent Trends in Data Type Specification. LNCS, vol. 1130, pp. 342–358. Springer, Heidelberg (1996)
4. Marre, B.: LOFT: a tool for assisting selection of test data sets from algebraic specifications. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 799–800. Springer, Heidelberg (1995)
5. Aiguier, M., Arnould, A., Boin, C., Le Gall, P., Marre, B.: Testing from algebraic specifications: test data set selection by unfolding axioms. In: Grieskamp, W., Weise, C. (eds.) FATES 2005. LNCS, vol. 3997, pp. 203–217. Springer, Heidelberg (2006)
6. Machado, P., Sannella, D.: Unit testing for Casl architectural specifications. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 506–518. Springer, Heidelberg (2002)

7. Arnould, A., Le Gall, P., Marre, B.: Dynamic testing from bounded data type specifications. In: Hlawiczka, A., Simoncini, L., Silva, J.G.S. (eds.) Dependable Computing - EDCC-2. LNCS, vol. 1150, pp. 285–302. Springer, Heidelberg (1996)
8. Aiguier, M., Arnould, A., Le Gall, P.: Exhaustive test sets for algebraic specification correctness. Technical report, IBISC - Université d'Évry-Val d'Essonne (2006)
9. Arnould, A., Le Gall, P.: Test de conformité: une approche algébrique. Technique et Science Informatiques, Test de logiciel 21, 1219–1242 (2002)
10. Mossakowski, T., Schröder, L., Roggenbach, M., Reichel, H.: Algebraic-coalgebraic specification in CoCasl. Journal of Logic and Algebraic Programming 67(1-2), 146–197 (2006)
11. Yannakakis, M., Lee, D.: Testing finite state machines. In: Symposium on Theory of Computing (STOC'91), pp. 476–485. ACM Press, New York (1991)
12. Tretmans, J.: Testing labelled transition systems with inputs and outputs. In: International Workshop on Protocols Test Systems (IWPTS'95) (1995)
13. Rusu, V., du Bousquet, L., Jéron, T.: An approach to symbolic test generation. In: Integrated Formal Methods (IFM '00), pp. 338–357. Springer, Heidelberg (2000)
14. Frantzen, L., Tretmans, J., Willemse, T.: Test generation based on symbolic specifications. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 1–15. Springer, Heidelberg (2005)
15. Gaston, C., Le Gall, P., Rapin, N., Touil, A.: Symbolic execution techniques for test purpose definition. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, pp. 1–18. Springer, Heidelberg (2006)
16. Ammann, P., Ding, W., Xu, D.: Using a model checker to test safety properties. In: International Conference on Engineering of Complex Computer Systems (ICECCS'01), pp. 212–221 (2001)
17. Aiguier, M., Arnould, A., Le Gall, P., Longuet, D.: Test selection criteria for quantifier-free first-order specifications. In: Fundamentals of Software Engineering (FSEN'07). Lecture Notes in Computer Science (to appear)
18. Longuet, D., Aiguier, M.: Specification-based testing for CoCasl's modal specifications. Technical report, IBISC - Université d'Évry-Val d'Essonne (2007) Available at `http://www.ibisc.fr/~dlonguet/publications_gb.html`
19. Bernot, G.: Testing against formal specifications: a theoretical view. In: Abramsky, S. (ed.) TAPSOFT 1991, CCPSD 1991, and ADC-Talks 1991. LNCS, vol. 494, pp. 99–119. Springer, Heidelberg (1991)
20. Hennicker, R., Wirsing, M., Bidoit, M.: Proof systems for structured specifications with observability operators. Theoretical Computer Science 173(2), 393–443 (1997)
21. Rutten, J.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249, 3–80 (2000)
22. Schröder, L., Mossakowski, T.: Coalgebraic modal logic in CoCasl. In: Recent Trends in Algebraic Specification Techniques (WADT'06). LNCS, vol. 4409, pp. 128–142 (2007)
23. Machado, P.: Testing from structured algebraic specifications. In: Rus, T. (ed.) AMAST 2000. LNCS, vol. 1816, pp. 529–544. Springer, Heidelberg (2000)