Test Selection Criteria for Modal Specifications of Reactive Systems*

Marc Aiguier Delphine Longuet

IBISC - CNRS FRE 2873 - University of Évry Val d'Essonne
523 place des terrasses de l'Agora - F91000 Évry
E-mail: {marc.aiguier,delphine.longuet}@ibisc.univ-evry.fr

Abstract

In the framework of functional testing from algebraic specifications, the strategy of test selection which has been widely and efficiently applied is based on axiom unfolding. In this paper, we propose to extend this selection strategy to a modal formalism used to specify dynamic and reactive systems. Such a work is then a first step to tackle testing of such systems more abstractly than most of the works dealing with what is called conformance testing. We get a higher level of abstraction since our specifications account for what is usually called underspecification, i.e. they do not denote a unique model but a class of models. Hence, the testing process can be applied at every design level.

Keywords. *Specification-based testing, dynamic specifications, selection criteria, unfolding, proof tree normalisation, exhaustivity, coalgebras*

Introduction

Specification-based testing is a particular case of black-box testing. It consists in executing the system under test on input data which have been selected from a specification. The aim is to show that the system behaviour conforms to its specification. Using formal specifications (i.e. specifications given as a formal text with a clear semantics) makes it possible to automate both test case generation from selection criteria and evaluation of test executions as successful or not. The evaluation of test executions, which consists in computing a success or failure verdict, is then done by comparing system outputs with the expected values defined by the specification. This often requires to make some restrictions on test cases so that they can be interpreted as successful or not when executed by the system under test. For instance, in the framework of testing from algebraic specifications, these restrictions consist in choosing as test cases formulae only built over Boolean connectives and ground equations whose equality predicate is defined within the programming language used to implement the system under test.

System correctness with respect to its specification is then defined up to these restrictions. This leads up to the notion of exhaustive set of test cases, whose successful submission to the system under test would prove its correctness. The problem is that the cardinality of an exhaustive set is usually too big (often infinite) to be fully submitted to the system. To manage the size of the exhaustive set, the idea is to make a partition of it, corresponding to the various cases described by the specification. Then, by assuming the standard uniformity hypothesis, meaning that each test case of each subdomain has the same power to make the system fail, one test case for each subdomain is submitted to the system. Hence, system correctness can be asymptotically reached by making an increasingly fine partition of the exhaustive set. This selection criterion leading to a partition of the exhaustive set has been mainly and extensively applied for specifications defined as theories of equational logic [1, 4, 5, 6, 7, 13] and more recently of quantifier-free first-order logic [2]. In these works, the partitioning of the exhaustive set under consideration is made in an algorithmic way by unfolding axioms. This axiom unfolding makes a case analysis of test purposes defined as simple equations in [1] and quantifier-free formulae in [2]. Test cases are then extracted from specifications by building input data defined by ground equations or formulae, matching the different cases defined in the specification.

In this paper, we propose to extend this selection criterion based on unfolding of axioms to dynamic and reactive aspects. Therefore, the formalism defined in this paper will manipulate modal formulae of the form $\bigwedge_{i\leq m}[m_i]t_i = t'_i \Rightarrow [m]t = t'$ where each m_i and m are modalities and t_i, t'_i, t, t' are terms whose function semantics depends on system states (see Section 1.1 for a complete definition of this modal formalism). It then is a simple extension of a pre-post language restricted to equations. Hence, we propose to

*This work has been performed within a French national project STACS (*Spécification et Test, Abstraits et Compositionnels, de Systèmes*) in collaboration with the French Atomic Energy Commission (CEA).

define a test selection method for testing dynamic systems.

Such a work is then a first step to tackle testing of dynamic and reactive systems more abstractly than most of the works dealing with the same subject [8, 9, 11, 17]. As a matter of fact, most of existing works deal with what is called conformance testing. This consists in showing that an implementation meets all the requirements of its specification when they are both represented by transition systems. The comparison is then made through behaviours, which are expressed as execution traces, i.e. sequences of possible actions. Here, we will go beyond by checking, as well as execution traces, properties on system attributes represented by equations. Moreover we get a higher level of abstraction, since specifications account for what is usually called *underspecification*, i.e. they do not denote unique systems but collections of systems. Therefore, the testing process can be applied at every design level. As far as we know, our approach is the first one that proposes to test dynamic and reactive systems with respect to a specification expressed in modal logic.

The paper is organised as follows. In Section 1, we define the formalism on which we will define the unfolding procedure. In order to be as self-contained as possible, Section 2 adapts relevant definitions of [13] to our framework of testing and defines selection criteria and their associated properties. We prove in this section the important result of the existence of an exhaustive test set, which allows to start the process of selecting test sets. Section 3 introduces the unfolding procedure allowing us to define a selection criterion for the class of specifications in our dynamic formalism. This unfolding procedure performs a case analysis on specification axioms defining the attributes (i.e. functions whose behaviour depends on system states) of the system under test. We will see that our unfolding procedure makes a strategy for selecting proof trees and then bounds the search space of proof trees. This is why it is definable in an algorithmic way. We will then show that this strategy coincides with the full derivability, i.e. 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.

1. Preliminaries

1.1. A dynamic formalism

A (dynamic) *signature* $\Sigma = (S, F, V)$ consists of a set S of sorts, a set F equipped with a partition F_d, F_a and F_m of function, attribute and method names, respectively, each one equipped with an arity in $S^* \times (S \cup \{\epsilon\})$[1], and an S-indexed set V of variables. In the sequel, a function, attribute or method f of arity $(s_1, \ldots, s_n, s)$ will be written $f : s_1 \times \ldots \times s_n \rightarrow s$. If $f \in F_d \cup F_a$, then $s \neq \epsilon$, otherwise $s = \epsilon$. Functions are operations on data, attributes are operations returning a value depending on the system state, and methods are operations making the system state evolve without returning any value. 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 (resp. the empty set) and function and attribute names in $F_d \cup F_a$ and preserving arity of operations. Finally, $M_\Sigma(V)$ and M_Σ are both sets of *modalities with variables* in V and *ground modalities* of form $m(t_1, \ldots, t_n)$ with $m : s_1 \times \ldots \times s_n \rightarrow$ in F_m and $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$ (resp. $(t_1, \ldots, t_n) \in T_{\Sigma s_1} \times \ldots \times T_{\Sigma s_n}$). A *substitution* is any mapping $\sigma : V \rightarrow T_\Sigma(V)$ that preserves sorts. Substitutions are naturally extended to terms with variables. Σ-*equations* are sentences of the form $t = t'$ where $t, t' \in T_\Sigma(V)_s$ for $s \in S$, and *modal formulae* are sentences of the form $[\alpha_1] \ldots [\alpha_n]\beta$ where $n \in \mathbb{N}$, for every i, $1 \leq i \leq n$, $\alpha_i \in M_\Sigma(V)$ and β is a Σ-equation. A positive conditional formula is then any sentence of the form $\varphi_1 \wedge \ldots \wedge \varphi_n \Rightarrow \varphi$ where for every i, $1 \leq i \leq n$, φ_i and φ are modal formulae or Σ-equations. $For(\Sigma)$ is the set of all positive conditional Σ-formulae. A *(positive conditional) specification* $Sp = (\Sigma, Ax)$ consists of a signature Σ and a set Ax of positive conditional formulae often called *axioms*.

Given a signature Σ, a Σ-model $\mathcal{M}$, so-called *Kripke frame*, is a triple $(\mathcal{W}, \mathcal{A}, \mathcal{R})$ where $\mathcal{W}$ is a non-empty set of states (or possible worlds), $\mathcal{A}$ is a $\mathcal{W}$-indexed family of algebras $\mathcal{A}^w$ over the algebraic signature $(S, F_d \cup F_a)$ such that for every $w, w' \in \mathcal{W}$, for every $s \in S$ and for every $f \in F_d$, $A^w_s = A^{w'}_s$ and $f^{\mathcal{A}^w} = f^{\mathcal{A}^{w'}}$, and $\mathcal{R}$ is a family of binary relations $\mathcal{R}_{m(v_1,\ldots,v_n)} \subseteq \mathcal{W} \times \mathcal{W}$, where $m : s_1 \times \ldots \times s_n \rightarrow \in F_m$ and $(v_1, \ldots, v_n) \in A_{s_1} \times \ldots \times A_{s_n}$, and such that for every $w \in \mathcal{W}$, the set $\{w' \mid w \, \mathcal{R}_{m(v_1,\ldots,v_n)} \, w'\}$ is finite.[2] $Mod(\Sigma)$ is the category objects of which are Σ-models. Given a Σ-model $\mathcal{M} = (\mathcal{W}, \mathcal{A}, \mathcal{R})$, we note $\_^A : T_{F_d} \rightarrow A$ the unique Σ-morphism that maps any ground term $f(t_1, \ldots, t_n)$ to $f^{\mathcal{A}}(t^A_1, \ldots, t^A_n)$.[3] A Σ-model $\mathcal{M}$ is said *reachable* if $\_^A$ is surjective. Given a Σ-model $\mathcal{M}$, a Σ-interpretation in $\mathcal{M}$ is any mapping $\iota : V \rightarrow A$ such that for every $s \in S$, $\iota(V_s) \subseteq A_s$. Interpretations are naturally extended to terms with variables: given an interpretation ι and a state $w \in \mathcal{W}$, we will note its extension $\iota_w : T_\Sigma(V) \rightarrow A$. $\mathcal{M}$ *satisfies* a Σ-equation $t = t'$ (resp. a modal formula $[\alpha_1] \ldots [\alpha_n]\beta$) for an interpretation ι and a state $w \in \mathcal{W}$, noted $\mathcal{M} \models_{\iota,w} t = t'$ (resp. $\mathcal{M} \models_{\iota,w} [\alpha_1] \ldots [\alpha_n]\beta$), if and only if $\iota_w(t) = \iota_w(t')$ (resp.

[1] ϵ stands for the empty word on S.

[2] Such Kripke frames are said image-finite (i.e. finitely branching). This condition is needed to get an exhaustive test set (see the proof of Theorem 1). However, it is not restrictive since, as we will see in Section 2, software systems will be assimilated to Kripke frames, and such a condition is sensible concerning systems.

[3] T_{F_d} is the restriction of T_Σ to functions of F_d.

for every $w' \in \mathcal{W}$ such that $w\mathcal{R}_{\iota_w(\alpha_1)} \bullet \ldots \bullet \mathcal{R}_{\iota_w(\alpha_n)}w'$,[4] $\mathcal{M} \models_{\iota,w'} \beta$). $\mathcal{M}$ *validates* a formula $\varphi_1 \wedge \ldots \wedge \varphi_n \Rightarrow \varphi$, noted $\mathcal{M} \models \varphi_1 \wedge \ldots \wedge \varphi_n \Rightarrow \varphi$, if and only if for every Σ-interpretation ι and every state $w \in \mathcal{W}$, if for every i, $1 \leq i \leq n$, $\mathcal{M} \models_{\iota,w} \varphi_i$ then $\mathcal{M} \models_{\iota,w} \varphi$. Given $\Psi \subseteq For(\Sigma)$ and two Σ-models $\mathcal{M}$ and $\mathcal{M}'$, $\mathcal{M}$ is Ψ-*equivalent to* $\mathcal{M}'$, noted $\mathcal{M} \equiv_\Psi \mathcal{M}'$, if and only if we have: $\forall \varphi \in \Psi$, $\mathcal{M} \models \varphi \iff \mathcal{M}' \models \varphi$. Given a specification $Sp = (\Sigma, Ax)$, a Σ-model $\mathcal{M}$ is an *Sp-model* if for every $\varphi \in Ax$, $\mathcal{M} \models \varphi$. $Mod(Sp)$ is the full subcategory of $Mod(\Sigma)$, objects of which are all Sp-models. A Σ-formula φ is a *semantic consequence* of a specification $Sp = (\Sigma, Ax)$, noted $Sp \models \varphi$, if and only if for every Sp-model $\mathcal{M}$, we have $\mathcal{M} \models \varphi$. $Sp^\bullet$ is the set of all semantic consequences.

A calculus for positive conditional specifications is defined by the following inference rules:

$$\frac{Ax \in Sp}{Sp \vdash Ax}\,\mathsf{Ax} \qquad \frac{}{Sp \vdash t=t}\,\mathsf{Ref} \qquad \frac{Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t=t'}{Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t'=t}\,\mathsf{Sym}$$

$$\frac{Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]t=t' \quad Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]t'=t''}{Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]t=t''}\,\mathsf{Trans}$$

$$\frac{Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]t_1=t'_1 \quad \ldots \quad Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]t_n=t'_n}{Sp \vdash \bigwedge_i \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]f(t_1,\ldots,t_n)=f(t'_1,\ldots,t'_n)}\,\mathsf{Cong}$$

$$\frac{Sp \vdash \bigwedge_{i \leq n} \varphi_i \Rightarrow \varphi}{Sp \vdash \bigwedge_i \sigma(\varphi_i) \Rightarrow \sigma(\varphi)}\,\mathsf{Subs} \qquad \frac{Sp \vdash \bigwedge_i \varphi_i \Rightarrow \varphi}{Sp \vdash \bigwedge_i \varphi_i \wedge \psi \Rightarrow \varphi}\,\mathsf{Mono}$$

$$\frac{Sp \vdash \bigwedge_i \varphi_i \Rightarrow \varphi}{Sp \vdash \bigwedge_i [\alpha]\varphi_i \Rightarrow [\alpha]\varphi}\,\mathsf{Nec} \qquad \frac{Sp \vdash \bigwedge_i \varphi_i \wedge \psi \Rightarrow \varphi \quad Sp \vdash \bigwedge_i \varphi_i \Rightarrow \psi}{Sp \vdash \bigwedge_i \varphi_i \Rightarrow \varphi}\,\mathsf{MP}$$

1.2. Running example

We take here as a running example a cash machine, or ATM, that allows customers to access their bank accounts in order to make cash withdrawals and to check their account balances. The customer first inserts his card, then verifies his identity by entering a passcode (PIN for Personal Identification Number). Upon successful entry of the PIN, the customer may perform a transaction, that is to check his account balance or to withdraw cash. If the number is entered incorrectly three times in a row, the card is not given back to the customer. If the customer asks for a withdrawal, he enters an amount that is checked not to go beyond the authorised threshold for this account. If it is the case, the withdrawal is not authorised, otherwise if the amount is available in the machine, the customer is given the money he asked for.

The signature of such a machine may then be the following:

$S = \{Nat, Bool\}$

$$\begin{array}{lll}
F_d = \{\neq & : Nat \times Nat \to Bool \\
\leq & : Nat \times Nat \to Bool \\
> & : Nat \times Nat \to Bool \\
+ & : Nat \times Nat \to Nat \\
- & : Nat \times Nat \to Nat \\
PIN & : Nat \to Nat \\
Balance & : Nat \to Nat \\
Threshold & : Nat \to Nat\}
\end{array}$$

$$\begin{array}{ll}
F_a = \{\mathsf{Card} :\to Nat & \mathsf{Code} :\to Nat \\
\mathsf{Amount} :\to Nat & \mathsf{Attempts} :\to Nat \\
\mathsf{Screen} :\to Nat & \mathsf{ATMamount} : Nat \to Bool
\end{array}$$

$$\begin{array}{ll}
F_m = \{\mathsf{card?} : Nat \to & \mathsf{passcode?} : Nat \to \\
\mathsf{check?} : \to & \mathsf{withdraw?} : \to \\
\mathsf{amount?} : Nat \to & \mathsf{cardback?} : \to \\
\mathsf{balance!} : \to & \mathsf{wrongcode!} : \to \\
\mathsf{cardkept!} : \to & \mathsf{notes!} : Nat \to \\
\mathsf{notenough!} : \to & \mathsf{threshold!} : \to
\end{array}$$

In function of a card number A, the ATM is able to know the passcode of the card $PIN(A)$, the account balance $Balance(A)$ and the maximum amount authorised for withdrawal $Threshold(A)$.

The state of the machine is known through six observators: Card gives the number of the inserted card if there is one, 0 otherwise; Code gives the entered code if a passcode has been entered, and 0 otherwise; Amount gives the asked amount if an amount has been entered, and 0 otherwise; ATMamount gives the total amount available in the machine; Attempts gives the number of wrong codes entered since a new card has been inserted; Screen gives the current display on the screen, 0 if nothing is displayed. [5].

Like in the setting of input output automata (IOLTS for instance), the state of the ATM evolves thanks to communications between the customer and the machine, that are emissions and receipts through channels. To keep the usual notations, although it does not have any effect on the semantics, method names denoting receipts will end with an interrogation mark '?', while method names denoting emissions will end with an exclamation mark '!'.

- Receipts, from the ATM point of view, are actions performed by the customer: the insertion of a card in the machine card?; the input of a code passcode?; the request for checking the account balance check?; the request for making a withdrawal withdraw?; the input of an amount to withdraw amount?; the request for getting the card back cardback?.

- Emissions are then actions performed by the machine, that are mainly messages to the customer, except the

[4] $\bullet$ is the composition of binary relations, and if $\alpha_i = m(t_1, \ldots, t_n)$, then $\iota_w(\alpha_i)$ denotes $m(\iota_w(t_1), \ldots, \iota_w(t_n))$.

[5] It is very simplified here since the screen can only display a natural number. The use of strings would just have made the example uselessly complicated.

issue of notes: balance! displays the account balance; wrongcode! tells the customer that the code he entered is refused; cardkept! swallows the card after three attempts to enter the right code; notes! gives the customer the money he asked for; notenough! says that the machine does not have money enough to give the customer the amount he wanted; threshold! tells the customer that he is not allowed to withdraw the amount he asked because it goes beyond his authorised threshold.

A specification of this ATM in our formalism may be the following.[6] Since we are interested in testing system dynamics, we only give here the axioms specifying attributes and methods, and suppose that functions of F_d have been specified separately, by using a classic algebraic formalism.

- $\mathsf{Card} = 0 \Rightarrow [\mathsf{card}?A]\ \mathsf{Card} = A$
- $\mathsf{Card} = A \wedge A \neq 0 \Rightarrow [\mathsf{card}?C]\ \mathsf{Card} = A$
- $[\mathsf{cardback}?]\ \mathsf{Card} = 0$
- $[\mathsf{cardkept}!]\ \mathsf{Card} = 0$
- $[\mathsf{card}?A]\ \mathsf{Card} = A \Rightarrow [\mathsf{card}?A]\ \mathsf{Code} = 0$
- $\mathsf{Code} = 0 \Rightarrow [\mathsf{passcode}?c]\ \mathsf{Code} = c$
- $\mathsf{Code} = c \wedge c \neq 0 \Rightarrow [\mathsf{passcode}?d]\ \mathsf{Code} = c$
- $[\mathsf{cardback}?]\ \mathsf{Code} = 0$
- $\mathsf{Code} \neq \mathit{PIN}(\mathsf{Card}) \Rightarrow [\mathsf{wrongcode}!]\ \mathsf{Code} = 0$
- $[\mathsf{cardkept}!]\ \mathsf{Code} = 0$
- $[\mathsf{withdraw}?]\ \mathsf{Amount} = 0$
- $[\mathsf{cardkept}!]\ \mathsf{Amount} = 0$
- $[\mathsf{card}?A]\ \mathsf{Card} = A \Rightarrow [\mathsf{card}?A]\ \mathsf{Amount} = 0$
- $\mathsf{Amount} = 0 \wedge \mathsf{Code} = \mathit{PIN}(\mathsf{Card}) \Rightarrow [\mathsf{amount}?M]\ \mathsf{Amount} = M$
- $\mathsf{Amount} = M \wedge M \neq 0 \Rightarrow [\mathsf{amount}?N]\ \mathsf{Amount} = M$
- $[\mathsf{cardback}?]\ \mathsf{Amount} = 0$
- $\mathsf{Amount} \leq \mathit{Threshold}(A) \wedge \mathsf{Amount} \leq \mathsf{ATMamount} \Rightarrow [\mathsf{notes}!]\ \mathsf{Amount} = 0$
- $\mathsf{Amount} \leq \mathit{Threshold}(A) \wedge \mathsf{Amount} > \mathsf{ATMamount} \Rightarrow [\mathsf{notenough}!]\ \mathsf{Amount} = 0$
- $\mathsf{Amount} > \mathit{Threshold}(A) \Rightarrow [\mathsf{threshold}!]\ \mathsf{Amount} = 0$
- $\mathsf{Amount} \leq \mathit{Threshold}(A) \wedge \mathsf{Amount} \leq \mathsf{ATMamount} \Rightarrow [\mathsf{notes}!]\ \mathsf{ATMamount} = \mathsf{ATMamount} - \mathsf{Amount}$
- $[\mathsf{card}?A]\ \mathsf{Card} = A \Rightarrow [\mathsf{card}?A]\ \mathsf{Attempts} = 0$
- $\mathsf{Code} \neq \mathit{PIN}(\mathsf{Card}) \Rightarrow [\mathsf{wrongcode}!]\ \mathsf{Attempts} = \mathsf{Attempts} + 1$
- $\mathsf{Attempts} > 2 \Rightarrow [\mathsf{cardkept}!]\ \mathsf{Attempts} = 0$
- $[\mathsf{cardback}?]\ \mathsf{Attempts} = 0$
- $[\mathsf{card}?A]\ \mathsf{Card} = A \Rightarrow [\mathsf{card}?A]\ \mathsf{Screen} = 0$
- $[\mathsf{withdraw}?]\ \mathsf{Screen} = 0$
- $\mathsf{Code} = \mathit{PIN}(\mathsf{Card}) \Rightarrow [\mathsf{check}?][\mathsf{balance}!]\ \mathsf{Screen} = \mathit{Balance}(\mathsf{Card})$

[6] Not to make the specification too heavy, boolean functions are used as predicates. Obviously, a formula like $c \neq 0$ would have to be written $(c \neq 0) = \mathit{true}$.

The specification axioms give, for each attribute, the actions that modify it. Not to make the specification too heavy, all axioms expressing that an attribute remains unchanged after some actions don't appear. In the case of the attribute Amount for example, there should be the following four additional axioms:

- $\mathsf{Amount} = M \Rightarrow [\mathsf{passcode}?]\mathsf{Amount} = M$
- $\mathsf{Amount} = M \Rightarrow [\mathsf{wrongcode}!]\mathsf{Amount} = M$
- $\mathsf{Amount} = M \Rightarrow [\mathsf{check}?]\mathsf{Amount} = M$
- $\mathsf{Amount} = M \Rightarrow [\mathsf{balance}!]\mathsf{Amount} = M$

2. Testing from formal specifications

The work presented in Section 3 comes within the general framework of testing from formal specifications defined in [13]. Here, we succinctly introduce this framework and we instantiate it to the formalism we have just defined in Section 1.1.

Following previous works [4, 5, 6, 7, 13], 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 [10], 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 2.1 (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 $\mathcal{M} \equiv_{Obs} P$.*[7]

A test set allowing to establish the system correctness is

[7] Equivalence of Σ-models with respect to a set of formulae is defined in Section 1.1.

said *exhaustive*. Formally, an exhaustive set is defined as follows:

Definition 2.2 (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*

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

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 (hence the usefulness of subclass $\mathcal{K}$ of systems in Definition 2.2), and on the chosen set of observable formulae. For example, we will need here to assume that the system under test is reachable, as well as *initial* for modal formulae occurring in premises of axioms of Ax. Roughly speaking, a system will be said initial for a modal formula if it behaves like the specification for this modal formula.

Definition 2.3 (Initiality) *Let $Sp = (\Sigma, Ax)$ be a specification. Let $\mathcal{S} = (\mathcal{W}, \mathcal{A}, \mathcal{R}) \in Mod(\Sigma)$ be a system. Let φ be a ground modal Σ-formula. $\mathcal{S}$ is* initial *on φ if and only if we have:*

$$\forall w \in \mathcal{W}, \mathcal{S} \models_w \varphi \Rightarrow \begin{cases} \exists \alpha_1, \ldots, \alpha_m \in M_\Sigma, \exists w' \in \mathcal{W}, w' \, \mathcal{R}_{\alpha_1} \bullet \ldots \bullet \mathcal{R}_{\alpha_m} \, w \\ \qquad\qquad \wedge \; Sp \models [\alpha_1] \ldots [\alpha_m] \varphi \end{cases}$$

In practice, φ is of the form $f(v_1, \ldots, v_n) = t$ where $f : s_1 \times \ldots \times s_n \rightarrow s$ is an attribute (e.g. Amount) which is necessarily provided with at least one "modifier" $m : s_1 \times \ldots \times s_n \times s \rightarrow$ (e.g. amount?). A state is then often reduced to the "product" of signature attributes. The semantics of $m(v_1, \ldots, v_n, v)$ then consists in modifying $\mathcal{A}^w$ and yielding the algebra $\mathcal{A}^{w'}$ defined like $\mathcal{A}^w$ except for $f(v_1^{\mathcal{A}^w}, \ldots, v_n^{\mathcal{A}^w})$. For example, after an action like amount?M, the only part of the state which is modified is the attribute Amount. Hence, if $\mathcal{S} \models_w f(v_1, \ldots, v_n) = v$ then we have $Sp \models [m(v_1, \ldots, v_n, v')] f(v_1, \ldots, v_n) = v$, where v' is a function of v. Besides, the formula $[m(v_1, \ldots, v_n, v')] f(v_1, \ldots, v_n) = v$ is often given like an axiom of the specification. We have for example the axiom Amount $= 0 \wedge$ Code $= PIN($Card$) \Rightarrow$ [amount?M] Amount $= M$.

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 testing process, we first have to show that $Sp^\bullet \cap Obs$ is exhaustive. As stated by the following theorem, we show that given a dynamic specification $Sp = (\Sigma, Ax)$, the set $Sp^\bullet \cap Obs$ is exhaustive for a certain class $\mathcal{K}$ of systems when Obs is the set of ground modal formulae $\{[\alpha_1] \ldots [\alpha_n] t = t' \mid \forall i, 1 \leq i \leq n, \alpha_i \in M_\Sigma \wedge t, t' \in T_\Sigma\}$.

Theorem 1 *Let $Sp = (\Sigma, Ax)$ be a specification. The test set $Sp^\bullet \cap Obs$ is exhaustive for any reachable system $\mathcal{S}$ initial on all ground instances of any modal formula which occurs in premises of axioms of Ax.*

Idea of the proof. It recently became clear that a great variety of state-based dynamic systems can be captured uniformly as coalgebras [12, 18]. To prove the above theorem, we then show that the semantics of our formalism can be defined in the coalgebra theory. By using classic results of this theory, we build a final coalgebra as a model of specification Sp and then show that it is elementary equivalent to the system $\mathcal{S}$, up to Obs.

The entire proof may be found in [3].

The challenge of testing then consists in managing (infinite) test sets. In practice, experts apply some selection criteria on a reference test set in order to extract a test set of sufficiently reasonable size to be submitted to the system. The underlying idea is that all test sets 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" postulates that any chosen value is equivalent to another one.

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. Hence, by assuming the uniformity hypothesis, all test cases in T_i are equivalent to reveal incorrect systems with respect to the fault model captured by 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*.

For instance, the selection criterion we will define in the sequel of this paper consists in splitting a test set into subsets according to specification axioms. If we come back to the ATM specification of Section 1.2, the attribute Screen is specified by three axioms. Testing a formula like Screen $= n$ consists in finding both input data and a state. Input data are given by a ground substitution to apply to the formula in order to submit it to the system, and the state is given by a path leading to a state where the formula has to be verified. These substitutions and paths have to bring into play at least once each of these three axioms. Therefore, the set of test cases associated to Screen $= n$, where n is a variable, can be split into three subsets:

1. The set of tests associated to the reinitialisation of the attribute after the insertion of a new card, that is, associated to the substitution $n \mapsto 0$ and the path $\mathsf{card?}A$, coming from the axiom $[\mathsf{card?}A]\ \mathsf{Card} = A \Rightarrow [\mathsf{card?}A]\ \mathsf{Screen} = 0$.

2. The set of tests associated to the reinitialisation of the attribute after the request for a withdrawal, that is, associated to the substitution $n \mapsto 0$ and the path $\mathsf{withdraw?}$, coming from the axiom $[\mathsf{withdraw?}]\ \mathsf{Screen} = 0$.

3. The set of tests associated to the display of the account balance after the request for it, that is, associated to the substitution $n \mapsto \mathit{Balance}(\mathsf{Card})$ and the path $\mathsf{check?balance!}$, coming from the axiom $\mathsf{Code} = \mathit{PIN}(\mathsf{Card}) \Rightarrow [\mathsf{check?}][\mathsf{balance!}]\ \mathsf{Screen} = \mathit{Balance}(\mathsf{Card})$.

The process can be pursued on the first and the third subsets defined, respectively, by formulae $[\mathsf{card?}A]\ \mathsf{Card} = A$ and $\mathsf{Code} = \mathit{PIN}(\mathsf{Card})$.

Definition 2.4 (Selection criterion) *A* selection criterion C *is a mapping* $\mathcal{P}(Sp^\bullet \cap Obs) \rightarrow \mathcal{P}(\mathcal{P}(Sp^\bullet \cap Obs))$.[8] *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:*

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

A selection criterion consists of a mapping that splits test sets into families of test sets. The selection criterion is satisfied as soon as the considered test set contains at least 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 2.5 (Properties) *Let* C *be a selection criterion and* T *be a test set.*

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

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 we capture all test cases up to the notion of equivalent test cases (i.e. no test is lost).

In the following, we will denote by $T(Sp)$ the exhaustive test set $Sp^\bullet \cap Obs$.

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

3. Test of attributes by axiom unfolding

In this section, we study the problem of test case selection for dynamic specifications, by adapting a selection criteria based on axiom unfolding which has been widely and efficiently applied in the algebraic specification setting [1, 2, 5, 6, 7].

3.1. Input domain of attribute

Here, we are going to define a test selection method aimed at making a partition of $T(Sp)$. This selection method takes inspiration from classic methods that split the input domain of each signature function [1]. Here, because we are interested in testing dynamic systems, the signature functions we are going to consider are attributes.[9]

Succinctly, for a dynamic signature $\Sigma = (S, F, V)$ and an attribute f (i.e. $f \in F_a$), our method consists in

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

First, we have to define what input domain and test set for attributes are.

The input domain of an attribute f is the projection of the reference test set $T(Sp)$ on f, that is, the subset of $T(Sp)$ dealing with f. Obviously, it is an exhaustive test set devoted to the test of f.

Definition 3.1 (Input domain) *Let* $f : s_1 \times \ldots \times s_n \rightarrow s \in F_a$ *be an attribute. The* input domain *of* f*, noted* $T(Sp)_{|_f}$*, is the set defined as follows:*

$$T(Sp)_{|_f} = \{\, [\rho(\alpha_1)] \ldots [\rho(\alpha_m)]\rho(f(u_1, \ldots, u_n)) = \rho(v) \mid \rho : V \rightarrow T_\Sigma, \ [\rho(\alpha_1)] \ldots [\rho(\alpha_m)]\rho(f(u_1, \ldots, u_n)) = \rho(v) \in Sp^\bullet \cap Obs \,\}$$

Note that $[\alpha_1] \ldots [\alpha_m] f(u_1, \ldots, u_n) = v$ may be any formula, not necessarily in $Sp^\bullet$.

Example 1 The input domain of the attribute Amount, for example, contains all possible ground modal formulae which are semantic consequences of the specification of Section 1.2 concerning Amount. For instance:

$\mathsf{Amount} = 20$
$[\mathsf{amount?}50]\ \mathsf{Amount} = 50$
$[\mathsf{amount?}10]\ \mathsf{Amount} = 20$
$[\mathsf{passcode?}5438][\mathsf{wrongcode!}][\mathsf{cardkept!}]\ \mathsf{Amount} = 0$
$[\mathsf{amount?}20][\mathsf{notes!}][\mathsf{cardback?}]\ \mathsf{Amount} = 0$

belong to the input domain of Amount. ◇

[9] We recall that the data part of the system has been specified by classic algebraic specifications. So, testing this part can be done by using selection methods defined in [1, 2].

As we will see in the next section, axiom unfolding makes a partition of the input domain, that is the initial test set, by replacing a modal formula $[\alpha_1]\dots[\alpha_m]f(u_1,\dots,u_n) = v$ with sets of modal formulae, called constraints. These constraints correspond to the premises of the various cases described by the specification, that are axioms whose conclusion is $[\gamma_1]\dots[\gamma_q][\alpha_1]\dots[\alpha_m]f(u_1,\dots,u_n) = v$ (up to some substitutions). By construction, if all the constraints of a set are satisfied, so is the modal formula $[\gamma_1]\dots[\gamma_q][\alpha_1]\dots[\alpha_m]f(u_1,\dots,u_n) = v$. Hence, test sets are subsets of the input domain, satisfying constraints.

Definition 3.2 (Test set) *Let $\mathcal{C}$ be a set of modal Σ-formulae called* Σ-constraints*. Let $f : s_1 \times \dots \times s_n \to s \in F_a$ be an attribute, and let $\Delta = \alpha_1,\dots,\alpha_m$ be a sequence of modal terms in $M_\Sigma(V)$. A* test set *for f with respect to $\mathcal{C}$ and Δ, noted $T_{f,(\mathcal{C},\Delta)}$, is the set of ground Σ-formulae defined by:*

$$T_{f,(\mathcal{C},\Delta)} = \{[\gamma_1]\dots[\gamma_q][\rho(\alpha_1)]\dots[\rho(\alpha_m)]f(\rho(u_1),...,\rho(u_n)) = \rho(v) \mid \gamma_1,\dots,\gamma_q \in M_\Sigma, \rho : V \to T_\Sigma, \forall \xi \in \mathcal{C}, Sp \models [\gamma_1]\dots[\gamma_q]\rho(\xi)\ \}$$

Note that the input domain of an attribute f can be seen as a test set with an empty path and $f(u_1,\dots,u_n) = v$ as the only constraint, that is $\mathcal{C} = \{f(u_1,\dots,u_n) = v\}$ and $\Delta = \_$.

Unlike in the algebraic specification setting where constraints are only equations, here, constraints are both modal formulae and a path, since formula satisfaction depends on states. The path then gives us a state from which the constraints are satisfied.

We will then observe in the following section that the unfolding procedure build, step by step, paths $[\gamma_1]\dots[\gamma_q][\rho(\alpha_1)]\dots[\rho(\alpha_m)]$ increasingly complete, allowing to put the system in states in which observations $f(\rho(u_1),\dots,\rho(u_n)) = \rho(v)$, where ρ is a ground substitution, are satisfied.

3.2. Unfolding procedure

The unfolding procedure inputs are:

- a positive conditional specification $Sp = (\Sigma, Ax)$,
- an attribute $f \in F_a$, and
- a set Γ of couples $(\mathcal{C}, \Delta)$ where $\mathcal{C}$ is a Σ-constraints set and Δ is a finite sequence of modal terms.

Test sets for attributes are naturally extended to sets of couples Γ as follows:

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

The first set Γ_0 contains the unique couple $(\mathcal{C}_0, \_)$ defined by $\mathcal{C}_0 = \{f(x_1,\dots,x_n) = y\}$ where for every i, $1 \le i \le n$, x_i and y are variables. "_" stands for the empty sequence of modal terms. Then

$$T_{f,\Gamma_0} = T_{f,(\{f(x_1,\dots,x_n)=y\},\_)} = T(Sp)_{|f}$$

is the initial test set.

This set will be split into test subsets thanks to the unfolding procedure, expressed by the two following inference rules:[10]

$$\textbf{Red}\ \frac{\Gamma \cup \{(\mathcal{C} \cup \{[\beta_1]\dots[\beta_k]t = r\}, \Delta)\}}{\Gamma \cup \{(\sigma(\mathcal{C}), \Delta)\}}\ \sigma \text{ mgu of } t \text{ and } r$$

$$\textbf{Unfold}\ \frac{\Gamma \cup \{(\mathcal{C} \cup \{\varphi\}\}, \Delta)\}}{\Gamma \cup \Big\{\Big(\bigcup_{c\in Tr(\varphi)} \{\mathcal{C}_c \cup c\}, \Delta_c\Big)\Big\}}$$

where:

- $Tr(\varphi)$ for $\varphi = [\beta_1]\dots[\beta_k]t = r$ is the set of Σ-constraint sets defined by:

$$\left\{ \begin{array}{l} \left\{ \begin{array}{c} [\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t[v]_\omega) = \sigma(r), \\ \sigma(\varphi_1),\dots,\sigma(\varphi_m) \end{array} \right\} \Bigg| \\ \Big(\bigwedge\limits_{1\le i\le m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]g(v_1,\dots,v_n) = v \in Ax \\ \qquad or \\ \bigwedge\limits_{1\le i\le m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]v = g(v_1,\dots,v_n) \in Ax \Big) \\ \sigma(t_{|\omega}) = \sigma(g(v_1,\dots,v_n)),\ and \\ p \ge k, \forall l, 1 \le l \le k, \sigma(\beta_l) = \sigma(\gamma_{(p-k)+l}),\ \sigma \text{ unifier} \end{array} \right\}$$

$$\bigcup$$

$$\left\{ \begin{array}{l} \left\{ \begin{array}{c} [\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t) = \sigma(r[v]_\omega), \\ \sigma(\varphi_1),\dots,\sigma(\varphi_m) \end{array} \right\} \Bigg| \\ \Big(\bigwedge\limits_{1\le i\le m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]g(v_1,\dots,v_n) = v \in Ax \\ \qquad or \\ \bigwedge\limits_{1\le i\le m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]v = g(v_1,\dots,v_n) \in Ax \Big) \\ \sigma(r_{|\omega}) = \sigma(g(v_1,\dots,v_n)),\ and \\ p \ge k, \forall l, 1 \le l \le k, \sigma(\beta_l) = \sigma(\gamma_{(p-k)+l}),\ \sigma \text{ unifier} \end{array} \right\}$$

- for every $\{[\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t[v]_\omega) = \sigma(r), \sigma(\varphi_1), \dots, \sigma(\varphi_m)\}$ (resp. $\{[\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t) = \sigma(r[v]_\omega), \sigma(\varphi_1),\dots,\sigma(\varphi_m)\}$) in $Tr(\varphi)$, $\mathcal{C}_c$ is the set $\{[\sigma(\gamma_1)]\dots[\sigma(\gamma_{p-k})][\sigma(\delta_1)]\dots[\sigma(\delta_q)]\sigma(\varepsilon) \mid [\delta_1)]\dots[\delta_q]\varepsilon \in \mathcal{C}\}$, and

[10] The most general unifier (or mgu) of two terms t and r is the most general substitution σ such that $\sigma(t) = \sigma(r)$.

- Δ_c is the sequence $\sigma(\gamma_1), \ldots, \sigma(\gamma_{p-k}), \alpha_1, \ldots, \alpha_q$ when $\Delta = \alpha_1, \ldots, \alpha_q$.

The definition of $Tr(\varphi)$ being based on the subterm relation and unification, this set is computable if the specification Sp has a finite set of axioms.

The **Red** rule eliminates tautologies from constraints sets. Intuitively, the **Unfold** rule consists in replacing the formula φ with a set c of constraints, which are what remains of the axiom after unification, and in lengthening the path Δ with the path $\gamma_1 \ldots \gamma_{p-k}$ given by the axiom.

Hence, given a modal formula φ, we have the selection criterion C_φ that maps any $T_{f,(\mathcal{C},\Delta)}$ to $(T_{f,(\mathcal{C}'_c \cup c, \Delta_c)})_{c \in Tr(\varphi)}$ if $\varphi \in \mathcal{C}$, $T_{f,(\mathcal{C},\Delta)}$ otherwise, where $\mathcal{C}' = \mathcal{C} \setminus \{\varphi\}$.

Example 2 We want to test the attribute Screen of the specification of Subsection 1.2. The initial test set for Screen is its input domain $T(Sp)_{|\mathsf{Screen}}$, that is the test set $T_{\mathsf{Screen},\Gamma_0}$ where $\Gamma_0 = \{(\mathcal{C}_0, \Delta_0)\}$ with $\mathcal{C}_0 = \{\mathsf{Screen} = n\}$ and $\Delta_0 = \_$, as explained above. Then by applying the **Unfold** rule, we obtain the set $\Gamma_1 = \{(\mathcal{C}_1, \Delta_1), (\mathcal{C}_2, \Delta_2), (\mathcal{C}_3, \Delta_3)\}$ where:

- $\mathcal{C}_1 = \{[\mathsf{card?}C]\ \mathsf{Card} = C, [\mathsf{card?}C]\ \mathsf{Screen} = 0\}$ and $\Delta_1 = \mathsf{card?}C$, which come from the unification with the axiom $[\mathsf{card?}A]\ \mathsf{Card} = A \Rightarrow [\mathsf{card?}A]\ \mathsf{Screen} = 0$.

- $\mathcal{C}_2 = \{[\mathsf{withdraw?}]\ \mathsf{Screen} = 0\}$ and $\Delta_2 = \mathsf{withdraw?}$, which come from the unification with $[\mathsf{withdraw?}]\ \mathsf{Screen} = 0$.

- $\mathcal{C}_3 = \{\mathsf{Code} = PIN(\mathsf{Card}), [\mathsf{check?}][\mathsf{balance!}]\ \mathsf{Screen} = Balance(\mathsf{Card})\}$ and $\Delta_3 = \mathsf{check?balance!}$, which come from the unification with $\mathsf{Code} = PIN(\mathsf{Card}) \Rightarrow [\mathsf{check?}][\mathsf{balance!}]\ \mathsf{Screen} = Balance(\mathsf{Card})$.

A second step of unfolding is then possible for constraints in $\mathcal{C}_1$ and in $\mathcal{C}_3$.

The unfolding of constraints in $\mathcal{C}_1$ leads to the set $\Gamma_2^1 = \{(\mathcal{C}'_1, \Delta'_1)\}$ where $\mathcal{C}'_1 = \{\mathsf{Card} = 0, [\mathsf{card?}C]\ \mathsf{Card} = C, [\mathsf{card?}C]\ \mathsf{Screen} = 0\}$ and $\Delta'_1 = \mathsf{card?}C$, which come from the unification of $[\mathsf{card?}C]\ \mathsf{Card} = C$ with the axiom $\mathsf{Card} = 0 \Rightarrow [\mathsf{card?}A]\ \mathsf{Card} = A$.

The unfolding of constraints in $\mathcal{C}_3$ leads to the set $\Gamma_2^3 = \{(\mathcal{C}'^1_3, \Delta'^1_3), (\mathcal{C}'^2_3, \Delta'^2_3)\}$ where:

- $\mathcal{C}'^1_3 = \{\mathsf{Code} = 0, [\mathsf{passcode?}PIN(\mathsf{Card})]\mathsf{Code} = PIN(\mathsf{Card}), [\mathsf{passcode?}PIN(\mathsf{Card})][\mathsf{check?}][\mathsf{balance!}]\ \mathsf{Screen} = Balance(\mathsf{Card})\}$ and $\Delta'^1_3 = \mathsf{passcode?}PIN(\mathsf{Card})\mathsf{check?balance!}$, which come from the unification of $\mathsf{Code} = PIN(\mathsf{Card})$ with the axiom $\mathsf{Code} = 0 \Rightarrow [\mathsf{passcode?}c]\ \mathsf{Code} = c$.

- $\mathcal{C}'^2_3 = \{\mathsf{Code} = PIN(\mathsf{Card}), PIN(\mathsf{Card}) \neq 0, [\mathsf{passcode?}p]\mathsf{Code} = PIN(\mathsf{Card}), [\mathsf{passcode?}p][\mathsf{check?}][\mathsf{balance!}]\mathsf{Screen} = Balance(\mathsf{Card})\}$ and $\Delta'^1_3 = \mathsf{passcode?}p\,\mathsf{check?balance!}$, which come from the unification of $\mathsf{Code} = PIN(\mathsf{Card})$ with the axiom $\mathsf{Code} = c \wedge c \neq 0 \Rightarrow [\mathsf{passcode?}d]\ \mathsf{Code} = c$.

The constraint $[\mathsf{withdraw?}]\ \mathsf{Screen} = 0$ of $\mathcal{C}_2$ can not be unified with any axiom, so the unfolding is finished for this set.

We observe that each step of the procedure builds paths increasingly long to reach states where $\mathsf{Screen} = n$ is satisfied, under constraints on initial states of paths.

As another example, if we wanted to test the attribute Attempts, the initial test set, or input domain of Attempts, is $T_{\mathsf{Attempts},\Gamma_0}$ where $\Gamma_0 = \{(\mathcal{C}_0, \Delta_0)\}$ with $\mathcal{C}_0 = \{\mathsf{Attempts} = n\}$ and $\Delta_0 = \_$. But Attempts can only take the values 0, 1 and 2. The unfolding of $\mathsf{Attempts} = n$ will then lead to two kinds of constraints: those where $n < 3$, that will become test cases since they are consequences of the specification, and those where $n \geq 3$ that will not lead to test cases. The unfolding procedure cannot distinguish between these two kinds of constraints, however, before being submitted to the system, a ground substitution ρ is applied to both elements of test sets and constraints. Since by definition, the resulting ground formulae have to be consequences of the specification, constraints where $n \geq 3$ will not be submitted as test cases to the system. ◇

We write $\Gamma \vdash_U \Gamma'$ to mean that Γ can be transformed into Γ' by applying **Red** or **Unfold**. An unfolding procedure is then a program that accepts in input a positive conditional specification Sp, and uses the above inference rules to generate a sequence

$$\Gamma_0 \vdash_U \Gamma_1 \vdash_U \Gamma_2 \ldots$$

Termination of the unfolding procedure is unlikely, since it is not checked, during its execution, whether the formula $[\beta_1] \ldots [\beta_k]t = r$ is a semantic consequence of the specification or not. Actually, this will be done during the selection phase, either automatically when the specification under consideration is decidable or "by hand" otherwise. The aim of the unfolding procedure is to make a partition of $T(Sp)$ increasingly fine. We can then observe that axiom unfolding defines a proof strategy which enables to bound the search space for proof trees (see the proof of Theorem 2). The idea is then to stretch further the execution of the procedure in order to make increasingly big proof trees whose remaining lemmas are constraints. If among remaining lemmas, some of them are not true, then the associated test set is empty.

3.3. Soundness and completeness

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

The completeness result needs the following additional assumption: for any Γ resulting of the unfolding procedure, any $(\mathcal{C}, \Delta) \in \Gamma$, any $\xi \in \mathcal{C}$ and any $\varphi \in Ax$, $Var(\xi) \cap Var(\varphi) = \emptyset$.[11] This is a very weak assumption since it suffices to rename variables at each iteration of the procedure to satisfy it.

Theorem 2 *If* $\Gamma \vdash_U \Gamma'$*, then* $T_{f,\Gamma} = T_{f,\Gamma'}$.

Idea of the proof. As explained just before, we can observe that our unfolding procedure defines a proof search strategy that enables to bound the search space to the class of proof trees having the following structure:

- no instance of transitivity occurs over instances of congruence, necessitation, substitution and modus ponens;
- no instance of modus ponens occurs over instances of congruence and substitution;
- no instance of congruence occurs over instances of substitution.

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 normalising. This last result is shown by defining a recursive path ordering on proof trees.

The entire proof may be found in [3].

Conclusion

In this paper, we have extended a selection criteria based on unfolding of formulae to a dynamic formalism. As in the algebraic specification setting, our unfolding procedure consists in dividing the input domain into subdomains and then in selecting test cases from each of these subdomains. We have then proved that this unfolding is complete, that is test cases are preserved at each step of the unfolding procedure. This last result has been obtained by showing that the full derivability coincides with the derivability restricted to the class of proof trees generated by the unfolding procedure. We have also proved that the set of ground modal formulae satisfied by a specification Sp is exhaustive for every reachable system which is initial. To show this property we have first translated the semantics of our formalism into the coalgebras theory, and then to take the advantage of the duality between algebraic and coalgebraic approaches to prove this property of exhaustiveness.

We still have ongoing research concerning the extension of this unfolding procedure for a larger class of specification dealing with more general formulae than positive conditional ones. Actually, our goal is to be able to propose a framework of black-box testing for CoCasl specifications [16]. Our goal is also to propose a framework of functional testing for specification including structuration primitives. This last work would take inspiration from [14, 15].

[11] $Var(\psi)$ is the set of all variables occurring in ψ.

References

[1] M. Aiguier, A. Arnould, C. Boin, P. Le Gall, and B. Marre. Testing from algebraic specifications: test data set selection by unfolding axioms. In W. Grieskamp and C. Weise, editors, *Formal Approaches to Testing of Software (FATES)*, volume 3997 of *Lecture Notes in Computer Science*, pages 203–217. Springer-Verlag, 2005.

[2] M. Aiguier, A. Arnould, P. Le Gall, and D. Longuet. Test selection criteria for quantifier-free first-order specifications. In *IPM International Symposium on Fundamentals of Software Engineering (FSEN)*, Lecture Notes in Computer Science, 2007. To appear.

[3] M. Aiguier and D. Longuet. Test selection criteria for modal specifications of reactive systems. Technical Report IBISC-RR-2007-02, Université d'Évry-Val d'Essonne, 2007.

[4] A. Arnould and P. Le Gall. Test de conformité : une approche algébrique. *Technique et Science Informatiques, Test de logiciel*, 21:1219–1242, 2002.

[5] A. Arnould, P. Le Gall, and B. Marre. Dynamic testing from bounded data type specifications. In *Dependable Computing - EDCC-2*, volume 1150 of *Lecture Notes in Computer Science*, pages 285–302, Taormina, Italy, Octobre 1996. Springer.

[6] G. Bernot. Testing against formal specifications: a theoretical view. In *TAPSOFT'91, International Joint Conference on the Theory and Practice of Software Development*, volume 494 of *Lecture Notes in Computer Science*, pages 99–119, Brighton UK, 1991. Springer.

[7] 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.

[8] C. Bigot, A. Faivre, J.-P. Gallois, A. Lapitre, D. Lugato, J.-Y. Pierron, and N. Rapin. Automatic test generation with AGATHA. In *TACAS*, pages 591–596, 2003.

[9] L. Frantzen, J. Tretmans, and T. A. Willemse. Test generation based on symbolic specifications. In J. Grabowski and B. Nielsen, editors, *FATES 2004*, number 3395 in Lecture Notes in Computer Science, pages 1–15. Springer-Verlag, 2005.

[10] R. Hennicker, M. Wirsing, and M. Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2):393–443, 1997.

[11] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva. Symbolic test selection based on approximate analysis. In *11th*

Int. Conference on Tools and Algorithms for tthe Construction and Analysis of Systems (TACAS), Edinburgh, Scotland, April 2005.

[12] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, Fakultät für Mathematik und Informatik, University of Munich, 2000.

[13] P. Le Gall and A. Arnould. Formal specification and test: correctness and oracle. In *11th WADT joint with the 9th general COMPASS workshop*, volume 1130 of *Lecture Notes in Computer Science*, pages 342–358. Springer, 1996.

[14] P. Machado. Testing from structured algebraic specifications. In *AMAST2000*, volume 1816 of *Lecture Notes in Computer Science*, pages 529–544, 2000.

[15] P. Machado and D. Sannella. Unit testing for CASL architectural specifications. In *Mathematical Foundations of Computer Science*, number 2420 in Lecture Notes in Computer Science, pages 506–518. Springer-Verlag, 2002.

[16] T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel. Algebraic-co-algebraic specification in CoCASL. *Journal of Logic and Algebraic Programming*, 67(1-2):146–197, 2006.

[17] V. Rusu, L. du Bousquet, and T. Jéron. An approach to symbolic test generation. In *IFM'00: Proceedings of the Second International Conference on Integrated Formal Methods*, pages 338–357, London, UK, 2000. Springer-Verlag.

[18] J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.