Integration Testing from Structured First-Order Specifications via Deduction Modulo

Delphine Longuet[1] and Marc Aiguier[2]

[1] Laboratoire Spécification et Vérification, ENS Cachan,
61 avenue du Président Wilson, F-94235 Cachan Cedex
delphine.longuet@lsv.ens-cachan.fr
[2] Laboratory of Mathematics Applied to Systems (MAS), École Centrale Paris,
Grande voie des vignes, F-92295 Châtenay-Malabry
marc.aiguier@ecp.fr

Abstract. Testing from first-order specifications has mainly been studied for flat specifications, that are specifications of a single software module. However, the specifications of large software systems are generally built out of small specifications of individual modules, by enriching their union. The aim of integration testing is to test the composition of modules assuming that they have previously been verified, i.e. assuming their correctness. One of the main method for the selection of test cases from first-order specifications, called axiom unfolding, is based on a proof search for the different instances of the property to be tested, thus allowing the coverage of this property. The idea here is to use deduction modulo as a proof system for structured first-order specifications in the context of integration testing, so as to take advantage of the knowledge of the correctness of the individual modules.

Testing is a very common practice in the software validation process. The principle of testing is to execute the software system on a subset of its possible inputs in order to detect failures. A failure is detected if the system behaves in a non-conformant way with respect to its specification.

The testing process is usually decomposed into three phases: the *selection* of a relevant subset of the set of all the possible inputs of the system, called a test set; the *submission* of this test set to the system; the *decision* of the success or the failure of the test set submission, called the *oracle problem*. We focus here on the selection phase, which is the crucial point for the relevance and the efficiency of the testing process. In the approach called black-box testing, tests are selected from a (formal or informal) specification of the system, without any knowledge about the implementation.

Our work follows the framework defined by Gaudel, Bernot and Marre [1], for testing from specifications expressed in a logical formalism. One approach to selection consists first in dividing an exhaustive test set into subsets, and then in choosing one test case in each of these subsets, thus building a finite test set which covers the initial exhaustive test set. One of the most studied selection method for testing from equational (and then first-order) specifications is known as *axiom unfolding* [1–4]. Its principle is to divide the initial exhaustive test set according to criteria derived from the axioms of the specification, using the well-known and efficient proof techniques associated to first-order logic.

Contribution. Test case selection from first-order specifications have mainly been studied for flat specifications (and then flat programs), that are specifications of a single software module. However, for the description of large systems, it is convenient to compose specifications in a modular way [5]. The specification of a large system is generally built from small specifications of individual modules, that are composed by making their union and enriching it with new features in order to get new (larger) specifications, that are themselves composed and so on. The aim of integration testing is to test the composition of modules, assuming that these modules have previously been tested and then are correct. The assumption here is that the system under test is structured according to the structuration of its specification.

Here, we propose to use the knowledge of the correctness of individual modules to make the test selection method based on axiom unfolding more efficient. Since the modules are correct (i.e. they have already been sufficiently tested or completely proved), it is reasonable to assume to have an executable and complete specification of these modules, either from which their implementations has been build or which would have been generated from their implementations. Our selection method being defined for first-order specifications, it is important for this executable specification to be written in first-order logic. Of course, in the case where the specification has to be generated from the implementation, the generation may be more or less easy according to the programming language used (imperative or functional), but this is the price to pay to make the selection method efficient by taking advantage of the specification structure. However, we can observe that the obtained specification is most often composed of (conditional) equations that can be oriented from left to right into confluent and terminating (conditional) rewrite rules, and of predicate definition formulas of the form $p(t_1, \ldots, t_n) \Leftrightarrow \varphi$, where φ is a quantifier-free formula, that can be oriented into confluent and terminating rewrite rules on propositions (see Section 2). We will then suppose to have, for each individual module, a confluent and terminating rewrite system that completely specifies its behaviour. To preserve the black-box aspect of the approach (the tester has no knowledge about the implementation of the system and its modules), we suppose that these executable and complete specifications of modules have been written beforehand by the programmer.

In order to make our selection method more efficient, we propose to use the deduction modulo proposed by Dowek, Hardin and Kirchner [6] as a proof system for structured specifications. Deduction modulo is a formalism introduced to separate computations from deductions in proofs by reasoning modulo a congruence on propositions, which is defined by a rewrite relation over first-order terms and propositions. The idea behind deduction modulo is to hide the computational part of the proof in the congruence, in order to focus on its deductive part. In the context of integration testing, the same idea can be used to focus the proof on the new features coming from the composition of modules, relying on the correct behaviour of these modules which is embedded in the congruence. It leads to shorter proofs which take advantage of the structuration of specifications, thus making the selection procedure more efficient.

Related Work. Testing from structured first-order specifications has already been studied in the framework of institutions. Machado's works deal with the oracle problem [7], that is, whether a finite and executable procedure can be defined for interpreting the re-

sults of tests. When dealing with structured specifications, problems arise in particular with the union of specifications. Since the same sort and operations may be introduced and specified in different modules, the union will be consistent only if the different specifications of the same operations are. Doche and Wiels define an extension of the notion of institution to take test cases into account [8]. They incrementally generate tests from structured specifications, generating tests from small specifications and composing them according to a push-out of specifications.

Both of these works aim at building a general test set for the whole structured specification, composing individual test sets obtained for each of its part. The structuration of the specification helps to incrementally build the test set but not to actually test the program in an incremental way. We are here interested in incrementally testing from a structured specification, basing the construction of a test set on the success of the previous ones. Moreover, from the selection point of view, none of the mentioned works propose any particular strategy, but the substitution of axiom variables for some arbitrarily chosen data.

Organisation of the Paper. We first recall standard definitions about structuration of specifications (Section 1) and deduction modulo (Section 2). Section 3 introduces the general framework for testing from logical specifications and gives the result of the existence of an exhaustive test set for quantifier-free first-order specifications. We also prove the existence of an exhaustive test set for structured first-order specifications, relying on the correctness of the smaller modules. We restrict to quantifier-free formulas since we showed in [9] that existential formulas are not testable. Testing a formula of the form $\exists x \varphi(x)$ actually comes down to exhibiting a witness value a such that $\varphi(a)$ is interpreted as true by the system. Of course, there is no general way to exhibit such a relevant value, but notice that surprisingly, exhibiting such a value would amount to simply prove the system with respect to the initial property. In Section 4, the selection method by means of selection criteria is presented. We develop in Section 5 our test selection method from structured first-order specifications, by unfolding axioms using deduction modulo. We give the algorithm of the procedure and prove the soundness and completeness of the method, i.e. the preservation of exhaustiveness through unfolding.

1 Structured First-Order Specifications

A multi-sorted first-order signature $\Sigma = (S, F, P, V)$ is composed of a set of sorts S, a set of operations F, a set of predicates P and a set of variables V over these sorts. $T_\Sigma(V)$ and T_Σ are both S-indexed sets of terms with variables in V and ground terms, respectively, freely generated from variables and operations in Σ and preserving arity of operations. A substitution is any mapping $\sigma : V \to T_\Sigma(V)$ that preserves sorts. Substitutions are naturally extended to terms with variables. *Formulas* (or *propositions*) are built as usual in first-order logic from atomic formulas $p(t_1, \ldots, t_n)$, where p is a predicate and $t_1, \ldots, t_n$ are first-order terms, and Boolean connectives. Here, we only consider quantifier-free formulas. As usual, variables of quantifier-free formulas are implicitly universally quantified. A formula over Σ is said *ground* if it does not contain variables. Let us denote $For(\Sigma)$ the set of all formulas over the signature Σ.

A model of a signature Σ is a first-order structure giving an interpretation to sorts, operations and predicates of Σ. $Mod(\Sigma)$ is the set of models of Σ. The satisfaction of a quantifier-free formula φ by a given model $\mathcal{M}$ of Σ is inductively defined on the structure of φ as usual and denoted by $\mathcal{M} \models \varphi$. Given a set of formulas Ψ over Σ and two models $\mathcal{M}$ and $\mathcal{M}'$ of Σ, we say that $\mathcal{M}$ is *Ψ-equivalent to $\mathcal{M}'$*, denoted by $\mathcal{M} \equiv_\Psi \mathcal{M}'$, if and only if for every formula φ in Ψ, $\mathcal{M} \models \varphi$ if and only if $\mathcal{M}' \models \varphi$.

Given a specification $Sp = (\Sigma, Ax)$, a model $\mathcal{M}$ of Σ is a *model of Sp* if $\mathcal{M}$ satisfies all the formulas in Ax. $Mod(Sp)$ is the subset of $Mod(\Sigma)$ whose elements are the models of Sp. A formula φ over Σ is a *semantic consequence* of Sp, denoted by $Sp \models \varphi$, if and only if every model $\mathcal{M}$ of Sp satisfies φ. $Sp^\bullet$ is the set of all the semantic consequences of Sp.

The semantics of a specification $Sp = (\Sigma, Ax)$ is given by its signature $Sig(Sp) = \Sigma$ and its class of models $[\![Sp]\!] = Mod(Sp)$. The specification building operators allow to write *basic* (flat) specifications, to make the *union* of two specifications and to *enrich* specifications with additional sorts, operation and/or predicate and axioms [5]. In general, small specifications are written, for instance specifying basic operations and predicates for a given sort (Booleans, naturals, lists…), then they are composed by the union operator, and finally enriched by new sorts, operations, predicates and axioms involving several of the initial specifications (empty list, list length, list of the divisors of a natural…). The union and enrichment operators are defined as follows.

$$\textbf{Basic} \qquad Sp = (\Sigma, Ax) \qquad\qquad \textbf{Union} \qquad Sp = Sp_1 \textbf{ union } Sp_2$$
$$Sig(Sp) = \Sigma \qquad\qquad Sig(Sp) = Sig(Sp_1) \cup Sig(Sp_2)$$
$$[\![Sp]\!] = Mod(Sp) \qquad\qquad [\![Sp]\!] = [\![Sp_1]\!] \cap [\![Sp_2]\!]$$

$$\textbf{Enrich}^{1} \qquad Sp = \textbf{enrich } Sp_1 \textbf{ by sorts } S_2,\ \textbf{ops } F_2,\ \textbf{preds } P_2,\ \textbf{axioms } Ax_2$$
$$Sig(Sp) = Sig(Sp_1) \cup (S_2, F_2, P_2)$$
$$[\![Sp]\!] = \{\mathcal{M} \in Mod(Sig(Sp)) \mid \mathcal{M}_{|_{Sig(Sp_1)}} \in [\![Sp_1]\!] \ \wedge\ \mathcal{M} \models Ax_2\}$$

2 Deduction Modulo

A term rewrite rule $l \to r$ is a pair of terms l, r such that all free variables of r appear in l. A term rewrite system is a set of term rewrite rules. A proposition rewrite rule $A \to P$ is a pair composed of an atomic proposition A and a proposition P, such that all free variables of P appear in A. A rewrite system $\mathcal{R}$ is a pair consisting of a term rewrite system and a proposition rewrite system. We denote by $P \to_\mathcal{R} Q$ the fact that P can be rewritten to Q in the rewrite system $\mathcal{R}$ in one step. $\mathcal{R}$ may be omitted if it is clear from the context. $\xrightarrow{+}_\mathcal{R}$ (resp. $\xrightarrow{*}_\mathcal{R}$) is the transitive (resp. reflexive transitive) closure of this rewrite relation. We denote by $\equiv_\mathcal{R}$ the congruence generated by $\mathcal{R}$.

In the context of integration testing, we consider a system built from modules composed by union and enrichment and we assume the correctness of these modules. As we already explained in the introduction, since each module is correct, we suppose to have the most concrete specification of each individual module. When expressed in

[1] $\mathcal{M}_{|_\Sigma}$ stands for the reduct of $\mathcal{M}$ over the signature Σ.

first-order logic, this concrete specification (most often) leads to a terminating and confluent rewrite system. We will assume that the behaviour of the module is modelled by this terminating and confluent rewrite system, where the behaviour of functions and predicates is defined by rewrite rules over first-order terms and quantifier-free formulas respectively. For instance, if we take the simple example of the greatest common divisor (gcd) whose possible implementation written in Caml is:

```
let rec gcd(x,y) = if y > x then gcd(y,x)
                   else if x mod y = 0 then y
                        else gcd(y,x mod y);;
```

we obtain the following specification:

$$y > x \Rightarrow \gcd(x, y) = \gcd(y, x)$$
$$\neg(y > x) \wedge x \bmod y = 0 \Rightarrow \gcd(x, y) = y$$
$$\neg(y > x) \wedge \neg(x \bmod y = 0) \Rightarrow \gcd(x, y) = \gcd(y, x \bmod y)$$
$$x > y \Leftrightarrow (\neg(x = 0) \wedge y = 0) \vee (\mathrm{pred}(x) > \mathrm{pred}(y))$$

This specification can obviously be transformed into a set of confluent and terminating (conditional) rewrite rules on terms and propositions. On the contrary, the specification from which this implementation of gcd has been tested would rather be:

$$x \bmod \gcd(x, y) = 0 \qquad x \bmod z = 0 \wedge y \bmod z = 0 \Rightarrow \gcd(x, y) \geq z$$
$$y \bmod \gcd(x, y) = 0$$

A congruence relation $\equiv$ over formulas is naturally induced by these rewrite rules. For instance, we have the following equivalences:

$$\neg\big(\gcd(2x + 1, 2) = \gcd(2x, 2)\big) \equiv \big(\neg \gcd(2, 2x + 1 \mod 2) = 2\big) \equiv \neg\big(1 = 2\big)$$

Using deduction modulo to guide the selection of test cases thus allows to internalise in the congruence the knowledge of the individual modules correctness, in order to focus the testing procedure on the new features of the system coming from the composition of modules.

In order to deal with structured specifications, we must ensure that termination and confluence of the rewrite systems underlying the individual modules are preserved through the union of these modules. It has been proved that these properties of (simple) termination and confluence are preserved for finite rewrite systems that are composable [10] (the rewrite rules in different systems defining the same operation are the same). This property of composability is reasonable in a testing framework, since it is natural to suppose that an operation or a predicate appearing in different modules comes from the same underlying module (used by these modules) and then is implemented in the same way in every module. From now on, we will assume that the rewrite systems underlying the modules are composable pairwise, so their union is also terminating and confluent.

The sequent calculus modulo extends the usual sequent calculus by allowing to work modulo the rewrite system $\mathcal{R}$. When the congruence $\equiv_{\mathcal{R}}$ is the identity, this sequent calculus collapses to the usual one. The sequent calculus modulo is as powerful

as the usual sequent calculus: it is shown in [6] that a formula is provable in the sequent calculus modulo if and only if it is provable in the usual sequent calculus using an appropriate set of axioms which are called compatible.

Here, the sequent calculus modulo is dedicated to the inference of quantifier-free formulas, so the rules for the introduction of quantifiers are omitted. Moreover, the rules associated to Boolean connectives are reversible. Since we assume that the rewrite system $\mathcal{R}$ is terminating and confluent, the rules for Booleans connectives can be used to transform any sequent $\vdash \varphi$, where φ is a quantifier-free formula, into a set of sequents $\Gamma_i \vdash \Delta_i$ where every formula in Γ_i and Δ_i is atomic. Such sequents will be called *normalised sequents*. This transformation is obtained from basic transformations defined as rewriting rules between elementary proof trees. We showed in [11] that for the sequent calculus associated to quantifier-free formulas, every proof tree can be transformed into a proof tree of same conclusion and such that Cut and Subs rules never occur under rule instances associated to Boolean connectives. This result states that every sequent is equivalent to a set of normalised sequents, which allows to deal with normalised sequents only. Therefore, in the following, we will suppose that the specification axioms are given under the form of normalised sequents. We present the sequent calculus modulo for normalised sequents, which is defined by the following rules where $\Gamma \vdash_{\mathcal{R}} \Delta$ is a sequent such that Γ and Δ are two multisets of first-order formulas.

$$\frac{}{\Gamma, P \vdash_{\mathcal{R}} \Delta, Q}\ \textsf{Taut}\ \text{ if } P \equiv_{\mathcal{R}} Q \qquad \frac{}{\vdash_{\mathcal{R}} P}\ \textsf{Axiom}\ \text{ if } \exists ax \in Ax \text{ such that } P \equiv_{\mathcal{R}} ax$$

$$\frac{\Gamma \vdash_{\mathcal{R}} \Delta}{\Gamma' \vdash_{\mathcal{R}} \Delta'}\ \textsf{Subs}\ \text{ if } \Gamma' \equiv_{\mathcal{R}} \sigma(\Gamma) \text{ and } \Delta' \equiv_{\mathcal{R}} \sigma(\Delta) \qquad \frac{\Gamma, P \vdash_{\mathcal{R}} \Delta \qquad \Gamma' \vdash_{\mathcal{R}} Q, \Delta'}{\Gamma, \Gamma' \vdash_{\mathcal{R}} \Delta, \Delta'}\ \textsf{Cut}\ \text{ if } P \equiv_{\mathcal{R}} Q$$

where for a multiset Γ, $\sigma(\Gamma)$ is the multiset $\{\sigma(\varphi) \mid \varphi \in \Gamma\}$.

It is possible to show that, when normalised sequents are transformed into formulas in clausal form, the cut and substitution rules can be combined to obtain the classical resolution rule (see [12] for more details). Actually, as we will see afterwards, this is the rule of resolution which is implemented in our unfolding algorithm. However, we use the sequent calculus since it makes the correctness proof of this algorithm easier (see Theorem 3). It is well-known that resolution is complete for quantifier-free formulas. Then it follows from the results of [6] that the resolution modulo as defined above is also complete. Since the resolution modulo is equivalent to the sequent calculus modulo restricted to normalised sequents, this calculus is complete. From now on, we will then speak about theorems and semantic consequences without making any difference.

Example 1. We give here a specification of rationals, built as an enrichment of a specification of naturals NAT. Rationals are defined as pairs of naturals and the comparison predicate *strictly less than* is defined as usual from the same predicate over naturals.

spec RAT =
  enrich NAT **by**
  type $Rat ::= \_/\_\,(Nat, Nat)$
  pred $\ll$: $Rat \times Rat$
  vars x, y, u, v: Nat
  • $x/s(y) \ll u/s(v) \Leftrightarrow x \times s(v) < u \times s(y)$
end

This axiom gives the two following normalised sequents:

(1) $x/s(y) \ll u/s(v) \vdash x \times s(v) < u \times s(y)$
(2) $x \times s(v) < u \times s(y) \vdash x/s(y) \ll u/s(v)$

The module implementing NAT can be defined by the following rewrite system:

$$\begin{array}{lll} x + 0 \to x & x \times 0 \to 0 & x < 0 \to \bot \\ x + s(y) \to s(x + y) & x \times s(y) \to x + x \times y & 0 < s(x) \to \top \\ & & s(x) < s(y) \to x < y \end{array}$$

3 Testing from Logical Specifications

From now on, we assume that the specification of the system to be tested is given as a (structured) first-order specification $Sp = (\Sigma, Ax)$. Following previous works [2, 1, 13], we make the two following assumptions. First, the behaviour of the system under test can be described as a first-order structure, sharing the same signature as its specification. The system under test is thus considered to be a Σ-model. Secondly, test cases can be expressed as quantifier-free first-order formulas over the signature Σ. Some observability constraints must be imposed so that the system is able to evaluate the formulas chosen to be test cases as true or false. Such formulas are called *observable*. Test cases being quantifier-free first-order formulas, they must not contain non-instantiated variables to be evaluated by the system. Therefore here, observable formulas are all ground formulas. We will denote by Obs the set of observable formulas.

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

Following an observational approach [14], a system will be considered as a correct implementation of its specification if, as a model, it cannot be distinguished from a model of the specification. Since the system can only be observed through the observable formulas it satisfies, it is required to be equivalent to a model of the specification up to this notion of observability.

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

The correctness of the system could then be proved if we were able to submit to the system the test set composed of all the observable formulas satisfied by the specification. Such a set is then said to be *exhaustive*.

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

The existence of an exhaustive test set ensures that for any incorrect system, there exists a test case making this system fail. To put it in a dual way, it ensures that it is

relevant to test this system with respect to its specification since its correctness can be asymptotically approached by submitting a potentially infinite test set. As a correctness reference, the exhaustive test set is then appropriate to start the selection of a finite test set of reasonable size. Note that, as we proved in [9], depending on the nature of the specification, on the observability restrictions and on the class of systems $\mathcal{K}$, an exhaustive test set does not necessarily exist.

Theorem 1 ([11]). *Let $Sp = (\Sigma, Ax)$ be a quantifier-free first-order specification and Obs be the set of ground first-order formulas. Then $Sp^\bullet \cap Obs$ is exhaustive for $Mod(\Sigma)$.*

In the context of integration testing, we consider a system built from the composition of individual modules which have already been proved to be correct. The specification Sp of this system is structured by the union and the enrichment of its modules specifications. Since these modules are correct, what remains to be tested are the new behaviours coming from their composition. These new behaviours are properties involving several modules in the case of a union or involving new sorts, operations or predicates in the case of an enrichment. They are properties that do not involve a module alone, i.e. formulas over the new signature that are not formulas of a module's signature alone: formulas in $For(\Sigma) \setminus (For(\Sigma_1) \cup For(\Sigma_2))$ if Σ is the union of Σ_1 and Σ_2; formulas in $For(\Sigma) \setminus For(\Sigma_1)$ if Σ is the enrichment of Σ_1. Let us denote $NewFor$ these sets of new formulas. Then, the new properties of the system coming from the composition are the formulas of $NewFor$ which are semantic consequences of the whole specification Sp. Let us denote $NewPr$ the set $NewFor \cap Sp^\bullet$. We have the following important result.

Theorem 2. *Let $Sp = \mathbf{enrich}\ Sp_1\ \mathbf{by}\ S_2, F_2, P_2, Ax_2$ (resp. $Sp = Sp_1\ \mathbf{union}\ Sp_2$). Let $\mathcal{K} = Mod(Sig(Sp))$. For every $\mathcal{S} \in \mathcal{K}$, if $Correct_{Obs}(\mathcal{S}_{|\Sigma_1}, Sp_1)$ (resp. and $Correct_{Obs}(\mathcal{S}_{|\Sigma_2}, Sp_2)$), then $NewPr \cap Obs$ is exhaustive for $\mathcal{S}$.*[3]

The proof may be found in the long version of this paper [15]. The key argument is that the behaviour of the modules is completely known, so their specifications are complete and $S_{|\Sigma_i}$ is fully characterised by the set of ground consequences of its specification Sp_i (it satisfies exactly all formulas of $Sp_i^\bullet \cap Obs$ and not any other). Therefore there are no new properties about the modules in $NewPr$, since the observability is the same.

4 Selection Criteria

When it exists, the exhaustive test set is the starting point for the selection of a practical test set. In practice, experts apply selection criteria on a reference test set in order to extract a test set of reasonable size to submit to the system. The aim is to divide the initial set according to a given criterion, in order to obtain subsets corresponding to particular behaviours representing this criterion. This selection method is called partition testing.

[3] $\mathcal{S}_{|\Sigma}$ stands for the reduct of $\mathcal{S}$ over the signature Σ.

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

For $C(T)$ a set of test sets T_i, we denote by $|C(T)|$ the set $\bigcup_i T_i$.

Different selection criteria may be applied one after the other to get a finer and finer partition of the initial test set. When the subdivision of the initial test set is fine enough according to the tester, the construction of a finite test set covering this partition remains to be done. This is the *generation* phase. Here, an important assumption is needed, which is called the *uniformity hypothesis*. It states that in each of the obtained subsets, test cases all are equivalent to make the system fail [1]. In other words, every test case in a subset is representative of the whole subset, with respect to the selection criterion that has been applied. It is then sufficient to choose one test case in each subset to cover the whole initial test set. The construction of a test set relevant to a selection criterion must benefit from the division obtained by the application of this criterion. Test cases must be chosen so as not to loose any of the cases captured by the criterion.

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

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

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

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

Definition 5 (Properties). *Let C be a selection criterion and T be a test set. C is* sound for *T if and only if $|C(T)| \subseteq T$. C is* complete for *T if and only if $|C(T)| \supseteq T$.*

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

5 Axiom Unfolding for Structured Specifications

We present here our method for defining relevant selection criteria in order to guide the final choice of the test cases. The method we follow is called axiom unfolding [1–4]

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

and is adapted here to structured specifications in the context of integration testing. It basically consists of a case analysis of a property to test with respect to the specification axioms. The application of the selection criterion defined by this case analysis allows to refine the initial test set associated to the property by characterising test subsets which respect given constraints on the input data.

5.1 Test Sets for Quantifier-Free First-Order Formulas

Since the exhaustive test set is the set

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

one way to divide it is to divide the test set $\{\rho(\varphi) \mid \rho : V \to T_\Sigma, \rho(\varphi) \in Sp^\bullet\}$ associated to each formula φ in $NewFor$, i.e. the set of all the ground instances of φ that are semantic consequences of Sp. The selection criteria we are going to define allow to divide a test set associated to a formula, we will explain at the end of this section how to actually cover the whole exhaustive test set $NewPr \cap Obs$.

Definition 6 (Test set for a formula). *Let $\varphi \in NewFor$ be a formula, called* test purpose*. The* test set for *φ, denoted by T_φ, is the following set:*

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

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

As we will see in the next subsection, the division of a test set associated to a formula will result in a set of test subsets, representing sets of particular instances of the initial formula. These instances, called constrained test purposes, are characterised by a substitution of the variables and a set of constraints.

Definition 7 (Constrained test set). *Let $\varphi \in NewFor$ be a formula. Let $\mathcal{C} \subseteq For(\Sigma)$ be a set of formulas called* constraints *and $\sigma : V \to T_\Sigma(V)$ be a substitution. A* test set for φ constrained by $\mathcal{C}$ and σ*, denoted by $T_{(\mathcal{C},\sigma),\varphi}$, is the following set:*

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

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

5.2 Unfolding Procedure

The aim of the procedure is to compute a selection criterion dividing the test set associated to an initial test purpose, using the specification axioms. Each step of the procedure returns a partition of the initial test set, where each subset is characterised by a constrained test purpose. These subsets can themselves be divided again and so on, until the tester is satisfied with the obtained partition.

The initial test purpose φ can be seen as the constrained test purpose $((\{\varphi\}, \mathrm{Id}), \varphi)$, or even $((\mathcal{C}_0, \mathrm{Id}), \varphi)$ where $\mathcal{C}_0$ is the set of normalised sequents obtained from φ. Let

Ψ_0 be the set containing the initial constraints of test purpose φ, the pair $(\mathcal{C}_0, \mathrm{Id})$. Constrained test sets for formulas are naturally extended to sets of pairs Ψ as follows: $T_{\Psi,\varphi} = \bigcup_{(\mathcal{C},\sigma)\in\Psi} T_{(\mathcal{C},\sigma),\varphi}$. The initial test set T_φ then is the set $T_{\Psi_0,\varphi}$.

The aim of the procedure is to divide this set according to the different cases in which formula φ holds. These cases correspond to the different instances of φ that can be proved as theorems. In the context of integration testing, the idea is to use the sequent calculus modulo presented in Section 2 to search for proofs of instances of φ relying on the correctness of the smaller modules of the implementation. So basically, the procedure searches for those proof trees that allow to deduce (instances of) the initial test purpose from the specification axioms modulo $\mathcal{R}$, where $\mathcal{R}$ is the rewrite system defined from the correct modules. However, the aim is not to build the complete proofs of these instances of φ, but only to make a partition of $T_{\Psi_0,\varphi}$ increasingly fine. A first step in the construction of the proof tree of each instance will give us pending lemmas, constraints remaining to prove that, together with the right substitution, characterise this instance of φ. We will thus be able to replace Ψ_0 with a set of constraints Ψ_1 characterising each instance of φ that can be proved from the axioms. The set Ψ_1 can itself be replaced by a bigger set Ψ_2 obtained from a second step in the construction of the previous proof trees, and so on. The procedure can be stopped at any moment, as soon as the tester is satisfied with the obtained partition.

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

To find a proof of an instance of φ, the procedure tries to unify φ with an axiom modulo $\mathcal{R}$. Only new axioms coming from an enrichment of a previous specification are considered here, since the behaviour of this previous specification is embedded in the congruence induced by $\mathcal{R}$. We denote this set of axioms $NewAx$ to avoid ambiguity. More precisely, it tries to unify a subset of the test purpose's subformulas with a subset of an axiom's subformulas, modulo $\mathcal{R}$. Hence, if the test purpose is a normalised sequent of the form

$$P_1, \ldots, P_p, \ldots, P_m \vdash Q_1, \ldots, Q_q, \ldots, Q_n$$

the procedure tries to unify a subset of $\{P_1, \ldots, P_m, Q_1, \ldots, Q_n\}$ with a subset of the formulas of an axiom. Then it looks for a specification axiom of the form

$$A_1, \ldots, A_p, A_{p+1}, \ldots, A_k \vdash B_1, \ldots, B_q, B_{q+1}, \ldots, B_l$$

such that it is possible to unify A_i and P_i modulo $\mathcal{R}$ for all i, $1 \leq i \leq p$, and to unify B_i and Q_i modulo $\mathcal{R}$ for all i, $1 \leq i \leq q$.

If the unification modulo with an axiom in $NewAx$ is possible, then the corresponding instance of the test purpose is provable from this axiom. Since A_i and P_i $(1 \leq i \leq p)$ on one hand and B_i and Q_i $(1 \leq i \leq q)$ on the other hand are unifiable modulo $\mathcal{R}$, there exists a substitution σ such that $\sigma(A_i) \equiv_\mathcal{R} \sigma(P_i)$ for all i, $1 \leq i \leq p$, and

such that $\sigma(B_i) \equiv_{\mathcal{R}} \sigma(Q_i)$ for all i, $1 \leq i \leq q$. Let us take the following notations: $\Lambda = \{A_1, \ldots, A_p\}$, $\Omega = \{B_1, \ldots, B_q\}$, $\Gamma = \{P_1, \ldots, P_p\}$, $\Gamma' = \{P_{p+1}, \ldots, P_m\}$, $\Delta = \{Q_1, \ldots, Q_q\}$, $\Delta' = \{Q_{q+1}, \ldots, Q_n\}$. We then get a proof tree of the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Lambda, C_1, \ldots, C_k \vdash \Omega, D_1, \ldots, D_l}\,\mathsf{Ax}}{\sigma(\Gamma), \sigma(C_1), \ldots, \sigma(C_k) \vdash \sigma(\Delta), \sigma(D_1), \ldots, \sigma(D_l)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\vdash R_1}}{\sigma(\Gamma), \sigma(C_2), \ldots, \sigma(C_k) \vdash \sigma(\Delta), \sigma(D_1), \ldots, \sigma(D_l)}\,\mathsf{Cut}}{\vdots}\,\mathsf{Cut} \quad \dfrac{\vdots}{\vdash R_k}}{\sigma(\Gamma), \sigma(C_k) \vdash \sigma(\Delta), \sigma(D_1), \ldots, \sigma(D_l)}\,\mathsf{Cut}}{S_1 \vdash \quad \sigma(\Gamma) \vdash \sigma(\Delta), \sigma(D_1), \ldots, \sigma(D_l)}\,\mathsf{Cut}}{\dfrac{\vdots}{\sigma(\Gamma'), S_l \vdash \sigma(\Delta')} \quad \dfrac{\vdots}{\sigma(\Gamma) \vdash \sigma(\Delta), \sigma(D_l)}\,\mathsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}$$

where $R_i \equiv_{\mathcal{R}} \sigma(C_i)$ for all i, $1 \leq i \leq k$ and where $S_i \equiv_{\mathcal{R}} \sigma(D_i)$ for all i, $1 \leq i \leq l$. The substitution σ together with the set of lemmas

$$c = \{ \vdash R_1, \ \ldots, \vdash R_k, \ S_1 \vdash, \ldots, \ \sigma(\Gamma'), S_l \vdash \sigma(\Delta') \}$$

characterise the instance of the test purpose φ derived from this proof tree, which corresponds to the constrained test purpose $((c, \sigma), \varphi)$.

Note that a priori, the lemmas R_i and S_i can be any formulas equivalent up to the congruence $\equiv_{\mathcal{R}}$. To avoid this non-determinism, we choose R_i and S_i in normal form: for all i, $1 \leq i \leq k$, R_i is the normal form of $\sigma(C_i)$ and for all i, $1 \leq i \leq l$, S_i is the normal form of $\sigma(D_i)$.

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

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

Unfold) As explained before, if a part of the constraint can be unified with a part of an axiom in $NewAx$ modulo $\mathcal{R}$, then we know that the constraint can be proved from this axiom with a certain number of applications of the Cut rule where each $\vdash R_i$ $(1 \leq i \leq k)$ and each $S_i \vdash$ $(1 \leq i \leq l)$ is a lemma remaining to prove. One of

those lemmas must bring the formulas of ψ not occurring in the axiom, so S_l is in the context $\sigma'(P_{p+1}), \ldots, \sigma'(P_m) \vdash S_l, \sigma'(Q_{q+1}), \ldots, \sigma'(Q_n)$.

Then the procedure replaces the initial constraint ψ with the sets of constraints in $Unf(\psi)$. Each unification with an axiom leads to a pair (c, σ'), so the initial constraint ψ is replaced with as many sets of formulas as there are axioms with which it can be unified. The definition of $Unf(\psi)$ being based on unification, this set is computable if the specification has finitely many axioms.

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

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

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

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

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

Example 2. We choose as a test purpose the formula $x < y \Rightarrow z/y \ll z/x$. The associated constrained test purpose for this formula is:

$$(\ (\ \{\ x < y \vdash z/y \ll z/x\ \},\ \mathrm{Id}\ ),\ x < y \Rightarrow z/y \ll z/x\ )$$

We denote by Ψ_0 the set containing this first pair of constraints. After a loop of the algorithm, we obtain a set Ψ_1 of constrained test purposes. To give a better intuition, we give the associated test subsets. The first set is obtained thanks to a unification of the test purpose with the left-hand side of axiom (2), and the second with the right-hand side of the same axiom.

$$\begin{array}{l} \{\ x \times s(v) < u \times s(y) \Rightarrow z/(u \times s(y)) \ll z/(x \times s(v)) \\ \qquad\qquad \mid x/s(y) \ll u/s(v) \Rightarrow z/(u \times s(y)) \ll z/(x \times s(v))\ \} \\ \{\ s(x) < s(y) \Rightarrow z/s(y) \ll z/s(x) \mid x < y \Rightarrow z \times s(x) < z \times s(y)\ \} \end{array}$$

Algorithm 1 Axiom unfolding

Inputs : structured quantifier-free first-order specification $Sp = (\Sigma, Ax)$,
rewrite system $\mathcal{R}$, test purpose $\varphi \in NewFor$
Output : set of constraints Ψ

$\Psi \leftarrow \{(C_0, \mathrm{Id})\}$ where C_0 is the set of normalised sequents obtained from φ
loop
Take (C, σ) from Ψ and remove it
Take $\psi = P_1, \ldots, P_m \vdash Q_1, \ldots, Q_n$ from C s.t. $\psi \in NewFor$ and remove it
$Unf(\psi) \leftarrow \emptyset$

(Reduce)
if there exists $\sigma' \in T_\Sigma(V)^V$ mgu, $1 \leq i \leq m$ and $1 \leq j \leq n$
such that $\sigma'(P_i) \equiv_\mathcal{R} \sigma'(Q_j)$ **then**
Add $(\emptyset, \sigma')$ to $Unf(\psi)$

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

(Unfold)
if ax is of the form $A_1, \ldots, A_p, C_1, \ldots, C_k \vdash B_1, \ldots, B_q, D_1, \ldots, D_l$
with $1 \leq p \leq m$, $1 \leq q \leq n$, and
there exists $\sigma' \in T_\Sigma(V)^V$ mgu such that
for all $1 \leq i \leq p$, $\sigma'(A_i) \equiv_\mathcal{R} \sigma'(P_i)$ and
for all $1 \leq i \leq q$, $\sigma'(B_i) \equiv_\mathcal{R} \sigma'(Q_i)$ **then**
$c \leftarrow \{\vdash R_i\}_{1 \leq i \leq k} \cup \{S_i \vdash\}_{1 \leq i \leq l-1}$
$\cup \{\sigma'(P_{p+1}), \ldots, \sigma'(P_m), S_l \vdash \sigma'(Q_{q+1}), \ldots, \sigma'(Q_n)\}$
such that $R_i = \sigma(C_i)\downarrow$ and $S_i = \sigma(D_i)\downarrow$
Add (c, σ') to $Unf(\psi)$

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

The premises of the constraint in the second subset is actually the normal form of the corresponding formula obtained after unification, which was $s(x) < s(y)$. Deduction modulo allows here to have a more concise proof, and then a more efficient selection procedure, thanks to the simplification allowed by the congruence.

5.3 Properties of the Selection Criterion

Here, we prove the two properties that make the unfolding procedure relevant for the selection of appropriate test cases, i.e. that the selection criterion defined by the procedure is sound and complete for the initial test set we defined. The entire proof may be found in [15].

Theorem 3 (Soundness and completeness). *Let φ be a quantifier-free first-order formula, $\mathcal{C}$ a set of constraints and $\sigma : V \to T_\Sigma(V)$ a substitution. Let $\psi \in \mathcal{C}$. The*

selection criterion for ψ is sound and complete *for the test set for φ constrained by $\mathcal{C}$ and σ:* $|C_\psi(T_{(\mathcal{C},\sigma),\varphi})| = T_{(\mathcal{C},\sigma),\varphi}$

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

To prove the completeness, we prove that all the possible instances of the test purpose can be proved with a proof tree of the form we showed earlier, and that the procedure generates all possible constraints for proving this instance. We thus prove that no test case is lost. Actually, we can observe that our unfolding procedure defines a proof search strategy that enables to limit the search space to the class of proof trees having the following structure: no instance of cut occurs over instances of substitution; there is no instance of cut whose premises both are instances of cut. We then have to prove that the derivability defined by our unfolding strategy coincides with the full derivability. To achieve this purpose, we 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.

6 Conclusion

In this paper, we investigated the problem of test case selection from structured specifications in the context of integration testing. The problem was to use the structuration of the specification as well as the unit testing result on the smaller modules of the system to select test cases allowing to test the new features of the system only, relying on the correctness of the modules. We used deduction modulo to guide the test case selection because it allows to easily integrate the knowledge of the correctness of the smaller modules in the rewrite system used as a congruence.

The definition of test selection criteria is the first step towards the construction of a practical test set to submit to the system. The next step is the generation of a test set satisfying these criteria. In our framework, the generation consists in applying the uniformity hypothesis to the constrained test sets obtained by unfolding an initial test purpose. It actually comes down to solve the constraints associated to each constrained test purpose, in order to build one test case corresponding to this purpose. Therefore, we plan to study the definition of an efficient algorithm of test case generation for (structured) quantifier-free first-order specifications.

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) (1991) 387–405
2. Bernot, G.: Testing against formal specifications: a theoretical view. In: Theory and Practice of Software Development. Volume 494 of LNCS. (1991) 99–119
3. Marre, B.: LOFT : a tool for assisting selection of test data sets from algebraic specifications. In: Theory and Practice of Software Development. Volume 915 of LNCS. (1995) 799–800

4. Aiguier, M., Arnould, A., Boin, C., Le Gall, P., Marre, B.: Testing from algebraic specifications: test data set selection by unfolding axioms. In: Formal Approaches to Testing of Software. Volume 3997 of LNCS. (2005) 203–217
5. Wirsing, M.: Algebraic specification. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Chapter 13. Elsevier (1990) 675–788
6. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. Journal of Automated Reasoning **31**(1) (2003) 33–72
7. Machado, P.: Testing from structured algebraic specifications. In: Algebraic Methodology and Software Technology. Volume 1816 of LNCS. (2000) 529–544
8. Doche, M., Wiels, V.: Extended institutions for testing. In: Algebraic Methodology and Software Technology. Volume 1816 of LNCS., Springer (2000) 514–528
9. Aiguier, M., Arnould, A., Le Gall, P., Longuet, D.: Exhaustive test sets for algebraic specification correctness. Technical report, IBISC, Université d'Évry (2008)
10. Ohlebush, E.: Modular properties of composable term rewriting systems. Journal of Symbolic Computation **20** (1995) 1–41
11. Aiguier, M., Arnould, A., Le Gall, P., Longuet, D.: Test selection criteria from quantifier-free first-order specifications. In: Fundamentals of Software Engineering. Volume 4767 of LNCS. (2007) 144–159
12. Longuet, D., Aiguier, M., Le Gall, P.: Proof-guided test selection from first-order specifications with equality. Journal of Automated Reasoning (2009) To appear.
13. Le Gall, P., Arnould, A.: Formal specification and test: correctness and oracle. In: 11th Workshop on Algebraic Development Techniques. Volume 1130 of LNCS. (1996) 342–358
14. Orejas, F., Navarro, M., Sánchez, A.: Implementation and behavioural equivalence : a survey. In: Recent Trends in Data Type Specification. Volume 655 of LNCS. (1993) 144–163
15. Longuet, D., Aiguier, M.: Integration testing from structured first-order specifications via deduction modulo. Technical report, LSV, ENS Cachan (2009)