Symbolic Execution Techniques for Test Purpose Definition

Christophe Gaston[1], Pascale Le Gall[2], Nicolas Rapin[1], and Assia Touil[2] ⋆

[1] CEA/LIST Saclay
F-91191 Gif sur Yvette, France
{christophe.gaston,nicolas.rapin}@cea.fr
[2] Université d'Évry, IBISC - FRE CNRS 2873
523 pl. des Terrasses F-91000 Évry, France
{legall,atouil}@lami.univ-evry.fr

Abstract. We propose an approach to test whether a system conforms to its specification given in terms of an Input/Output Symbolic Transition System (IOSTS). IOSTSs use data types to enrich transitions with data-based messages and guards depending on state variables. We use symbolic execution techniques both to extract IOSTS behaviours to be tested in the role of test purposes and to ground an algorithm of test case generation. Thus, contrarily to some already existing approaches, our test purposes are directly expressed as symbolic execution paths of the specification. They are finite symbolic subtrees of its symbolic execution. Finally, we give coverage criteria and demonstrate our approach on a running example.

Keywords: Conformance testing, Input/Output Symbolic Transition Systems, Test Purposes, Symbolic Execution, Coverage Criteria.

1 Introduction

Symbolic Transition Systems (STS) are composed of a data part and of a state-transition graph part. They specify behaviours of reactive systems with some benefits compared to the use of classical labelled transition systems. Models are often smaller and it is even possible to finitely denote systems having an infinite number of states. In this paper, following the works of [5,11,3], we are interested in studying conformance testing in the context of Input/Output Symbolic Transition Systems (IOSTS).

Approaches based on symbolic transformations make possible to exploit a particular analysis technique, the so-called *symbolic execution* [2,6], to define a test selection strategy. This technique has been first defined to compute program executions according to some constraints expressed on input values. The main idea is to use symbols instead of concrete data as input values and to derive a symbolic execution tree in order to describe all possible computations in a symbolic way. In our contribution, *test purposes* are defined as some particular

⋆ This work was partially supported by the RNRT French project STACS.

subtrees of this symbolic execution tree. They may be chosen by the user but we also propose criteria to automatically compute tests purposes. This is a response to industrial needs where engineers are not always able to define which behaviours they want to test. We introduce two criteria. The first one is called *all symbolic behaviours of length n* criterion. The second one is called *the restriction by inclusion* criterion: the extracted subtree satisfies a coverage criterion which is based on a procedure of redundancy detection. According to these test purposes, test cases are generated. Our algorithm for test case generation is given by a set of inference rules. Each rule is dedicated to handle an observation from the system under test (SUT) or a stimulation sent by the test case to the SUT. This testing process leads to a verdict being either *PASS*, *FAIL*, *INCONC* or *WeakPASS*. *PASS* means that the SUT succeeded in passing a test. *FAIL* means that a non-conformance has been detected. *INCONC* means that conformance is observed but the test purpose is not achieved while *WeakPASS* means that we are not sure to have achieved the test purpose. This last case is essentially due to the fact that the specifications may be non-deterministic.

Our work on symbolic conformance testing is close to the ones of [3,5]. Our contribution on generation of test cases is inspired by the one of [3]. But as the data part in [3] was only given according to a pure and abstract theoretical description, the implementation counterpart and examples are clearly missing. Associating a verdict to a test case execution requires to perform reachability analysis. Indeed, one must be able to compute as soon as possible whether or not a conformance may still be observed. In [5] over-approximation mechanisms based on abstract interpretation are used to perform reachability analysis. In our approach, we do not use such abstract interpretation techniques which have the drawbacks of both being difficult to use and of sometimes giving only approximated verdicts. We prefer to use symbolic execution based mechanisms which have been already successfully advocated in [10] to validate IOSTS models by exhibiting pertinent scenarios or deadlock situations.

The paper is structured as follow. In Section 2 we present the IOSTS formalism. Symbolic execution and restriction by inclusion are defined in Section 3. In Section 4, we present on-the-fly rules for generating test cases. In Section 5, we discuss the usage of coverage criterion for test purposes definition.

2 Input Output Symbolic Transition Systems

Input/Output Symbolic Transition Systems (IOSTS) extend Input Output Labelled Transition Systems (*IOLTS*) [10] by including data types. IOSTS are used to specify dynamic aspects. This is done by describing modifications of values associated to some variables, called *attribute variables*, in order to denote system state modifications. These modifications may be due to internal operations denoted by attribute variable substitutions or to interactions with the environment under the form of exchanges through communication channels of input/output messages. Those modifications may be conditioned by guards.

2.1 Data Types

Data types are specified with a *typed equational* specification framework.

Syntax A data type signature is a couple $\Omega = (S, Op)$ where S is a set of type names, Op is a set of operation names, each one provided with a profile $s_1 \cdots s_{n-1} \to s_n$ (for $i \leq n$, $s_i \in S$). Let $V = \bigcup_{s \in S} V_s$ be a set of typed variable names. The set of *Ω-terms* with variables in V is denoted $T_\Omega(V) = \bigcup_{s \in S} T_\Omega(V)_s$ and is inductively defined as usual over Op and V. $T_\Omega(\emptyset)$ is simply denoted T_Ω.

A *Ω-substitution* is a function $\sigma : V \to T_\Omega(V)$ preserving types. In the following, one notes $T_\Omega(V)^V$ the set of all the Ω-substitutions of the variables V. Any substitution σ may be canonically extended to terms.

The set $Sen_\Omega(V)$ of all typed equational *Ω-formulae* contains the truth values $true$, $false$ and all formulae built using the equality predicates $t = t'$ for $t, t' \in T_\Omega(V)_s$, and the usual connectives $\neg, \vee, \wedge, \Rightarrow$.

Semantics A *Ω-model* is a family $M = \{M_s\}_{s \in S}$ with, for each $f : s_1 \cdots s_n \to s \in Op$, a function $f_M : M_{s_1} \times \cdots \times M_{s_n} \to M_s$. We define *$\Omega$-interpretations* as applications ν from V to M preserving types, extended to terms in $T_\Omega(V)$. A model M satisfies a formula φ, denoted by $M \models \varphi$, if and only if, for all interpretations ν, $M \models_\nu \varphi$, where $M \models_\nu t = t'$ is defined by $\nu(t) = \nu(t')$, and where the truth values and the connectives are handled as usual. M^V is the set of all Ω-interpretations of V in M. Given a model M and a formula φ, φ is said *satisfiable* in M, if there exists an interpretation ν such that $M \models_\nu \varphi$.

In the sequel, we suppose that data types of our IOSTS correspond to the generic signature $\Omega = (S, Op)$ and are interpreted in a fixed model M. In the following, elements of M are called *concrete data* and denoted by terms of T_Ω.

2.2 Input/Output Symbolic Transition Systems

Definition 1 (*IOSTS*-signature). *An IOSTS-signature Σ is a triple (Ω, A, C) where Ω is a data type signature, $A = \bigcup_{s \in S} A_s$ is a set of variable names called* attribute variables *and C is a set of* communication channel *names.*

An IOSTS communicates with its environment through communication actions:

Definition 2 (Actions). *The* set of communication actions*, denoted $Act(\Sigma) = Input(\Sigma) \cup Output(\Sigma)$, is defined as follows, with $c \in C$, $y \in A$ and $t \in T_\Omega(A)$:*

$$Input(\Sigma) = \ c?y \mid c? \qquad \textit{and} \qquad Output(\Sigma) = \ c!t \mid c!$$

Elements of $Input(\Sigma)$ are stimulations of the system from the environment: $c?x$ (resp. $c?$) means that the system waits on the channel c for a value that will be assigned to the attribute variable x (resp. for a signal, for example, a pressed button). $Output(\Sigma)$ are responses of the system to the environment: $c!t$ (resp. $c!$) is the emission of the value t (resp. of a message without any sensible argument) through the channel c.

Definition 3 (*IOSTS*). *An IOSTS over Σ is a triple $G = (Q, q_0, Trans)$ where Q is a set of* state names, *$q_0 \in Q$ is the* initial state *and $Trans \subseteq Q \times Act_\Sigma(A) \times Sen_\Omega(A) \times T_\Omega(A)^A \times Q$. A transition (q, act, ϕ, ρ, q') of $Trans$ is composed of a source state q, an action act, a guard φ, a substitution of variables ρ and a target state q'. For each state $q \in Q$, there is a finite number of transitions of source state q.*

Observations for a communicating system are made of output actions. However, a system cannot always emit an output message from a given state q. It is then said to be *quiescent* [13]. In particular, quiescence from q depends on the current values of the attribute variables and on the guards of all transitions outgoing from q. As in [13], we can complete an IOSTS to explicit quiescent situations. For that, we add a special output communication action $\delta!$, expressing the absence of output, whose guard is complementary to all other guards of output transitions from q. This *enrichment by quiescence* is given by:

Definition 4 (Enrichment by quiescence). *Let $G = (Q, q_0, Trans)$ be an IOSTS over $\Sigma = (\Omega, A, C)$.* The enrichment of G by quiescence *is the IOSTS over $\Sigma_\delta = (\Omega, A, C \cup \{\delta\})$, defined by $G_\delta = (Q \cup \{q_\delta\}, q_0, Trans \cup Trans_\delta)$ where $(q, act, \varphi, \rho, q') \in Trans_\delta$ iff:*

- *$act = \delta!$, ρ is the identity substitution and $q' = q_\delta$.*
- *Let us note $tr_1, \cdots, tr_n$ all transitions of the form $tr_i = (q, act_i, \varphi_i, \rho_i, q_i)$ with $act_i \in Output(\Sigma)$. Then φ is $\wedge_{i \leq n} \neg \varphi_i$ if $n > 0$ and is true otherwise*[3].

Example 1. Let us consider an ATM system built over the communicating automaton depicted in Figure 1. This IOSTS specifies a system of cash withdrawal, with the initial state q_0. The user asks for some amount ($amount?x$). The ATM system checks if there is enough money in the account user (represented by the variable m) and if this is the first or the second time that the user withdraws money after a deposit. Then the user receives the asked amount by the channel *cash*. If the user account is less than 1000 then the withdrawal operation is not free and costs 1. Else, if there is not enough money in the account, the user receives an error message by the channel *screen*. The user can also deposit some money (t) in his bank account by the channel *deposit*. This is added to the bank account ($m := m + t$). Moreover, the user can ask for the amount of its account by the channel *check*, and receives the answer by the channel *sum*. There is only one transition labelled by $\delta!$ starting from the state q_0. Indeed, the state q_1 and q_2 are such that whatever the values of the attribute variables are, it is always possible to emit at least a message.

[3] If $\wedge_{i \leq n} \neg \varphi_i$ is not a satisfiable formula, the $(q, act, \wedge_{i \leq n} \neg \varphi_i, \rho, q')$ transition may clearly be omitted.

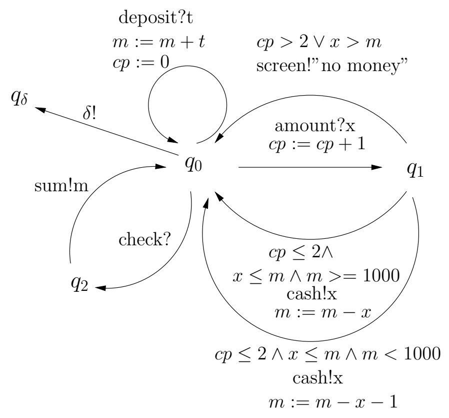


Fig. 1. Example of ATM system with withdrawal according to some conditions

2.3 Semantics

Definition 5 (Runs of a transition). *Let $tr = (q, act, \varphi, \rho, q') \in Trans$. Let us note $Act(M) = (C \times \{?, !\} \times M) \cup (C \times \{?, !\})$. The set $Run(tr) \subseteq M^A \times Act(M) \times M^A$ of* execution runs *of tr is such that $(\nu^i, act_M, \nu^f) \in Run(tr)$ iff:*

- *if act is of the form $c!t$ (resp. $c!$) then $M \models_{\nu^i} \varphi$, $\nu^f = \nu^i \circ \rho$ and $act_M = (c, !, \nu^i(t))$ (resp. $act_M = (c, !)$),*
- *if act is of the form $c?y$ then $M \models_{\nu^i} \varphi$, there exists ν^a such that $\nu^a(z) = \nu^i(z)$ for all $z \neq y$, $\nu^f = \nu^a \circ \rho$ and $act_M = (c, ?, \nu^a(y))$.*
- *if act is of the form $c?$ then $M \models_{\nu^i} \varphi$, $\nu^f = \nu^i \circ \rho$ and $act_M = (c, ?)$.*

We denote $source(tr)$ (resp. $target(tr)$) the source (resp. target) state q (resp. q') and $act(tr)$ stands for act. For a run $r = (\nu^i, act_M, \nu^f)$, we denote $source(r)$, $act(r)$ and $target(r)$ respectively ν^i, act_M and ν^f.

Definition 6 (Finite Paths of an *IOSTS*). *The set of finite paths in G, denoted $FP(G)$ contains all finite sequence $tr_1 \ldots tr_n$ of transitions in $Trans$ such that $source(tr_1) = q_0$ and for all $i < n$, $target(tr_i) = source(tr_{i+1})$.*

The runs of a finite path $tr_1 \ldots tr_n$ in $FP(G)$ are sequences $r_1 \ldots r_n$ such that for all $i \leq n$, r_i is a run of tr_i and for all $i < n$, $target(r_i) = source(r_{i+1})$.

The set of concrete traces of a finite path *$p = tr_1 \ldots tr_n$, denoted $Trace(p)$ is the set of finite action sequences $act(r_1) \ldots act(r_n)$ for any run $r_1 \cdots r_n$ of p.*

In the following, and as usual, for any $p \in FP(G)$, $length(p)$ denotes the number of occurrences of the transitions in the definition of p. We also note $Ext_{out}(p)$ the set of finite paths of G extending p by a transition introducing an output action. Formally, $Ext_{out}(p) = \{p' \in FP(G) \mid p' = p.tr \wedge act(tr) \in Output(\Sigma)\}$.

Definition 7. *The* semantics *of an IOSTS G is* $Trace(G) = \bigcup_{p \in FP(G)} Trace(p)$

3 Symbolic Execution

3.1 Definition

In our context, we call a *symbolic behaviour* of an IOSTS any finite path p of this IOSTS for which $Trace(p) \neq \emptyset$. In order to characterize the set of traces of a symbolic behaviour we propose to use a *symbolic execution* mechanism. Symbolic execution has been first defined for programs [6,2,9]. This technique can naturally be adapted to the framework of IOSTS. The main idea is to replace concrete input values and initialization values of attribute variables by symbolic ones with fresh variables and to compute the constraints on these variables: those constraints are called *path conditions*. In the sequel we assume that those fresh variables are chosen in a set $F = \bigcup_{s \in S} F_s$ disjoint from the set of attribute variables A. We now give the intermediate definition of *symbolic extended state* which is a structure allowing to store information about a symbolic behaviour: the IOSTS current state (target state of the last transition of the symbolic behaviour), the path condition and the symbolic values associated to attribute variables.

Definition 8 (Symbolic extended state). *A* symbolic extended state over *F for an IOSTS $G = (Q, q_0, Trans)$ is a triple $\eta = (q, \pi, \sigma)$ where $q \in Q$, $\pi \in Sen_{\Omega}(F)$ is called a* path condition *and $\sigma \in T_{\Omega}(F)^A$. $\eta = (q, \pi, \sigma)$ is said to be satisfiable if π is satisfiable*[4]*. One notes $\mathcal{S}$ (resp. $\mathcal{S}_{sat}$) the set of all the (resp. satisfiable) symbolic extended states over F.*

We now define the symbolic execution of an IOSTS. Intuitively, the symbolic execution of an IOSTS can be seen as a tree whose edges are symbolic extended states and vertexes are labelled by symbolic communication actions. The root is a symbolic extended state made of the IOSTS initial state, the path condition *true* (there is no constraint to begin the execution) and of an arbitrary initialization σ_0 of variables of A in F. Vertexes are computed by choosing a source symbolic state η already computed and by symbolically executing a transition of the IOSTS whose source is the state introduced in η. The symbolic communication action is computed from the transition communication action and from the symbolic values associated to attribute variables in η. A target symbolic extended state is then computed. It stores the target state of the transition, a new path condition derived from the path condition of η and from the transition guard, and finally the new symbolic values associated to attribute variables.

Definition 9 (Symbolic execution of an IOSTS). *Let $G = (Q, q_0, Trans)$ be an IOSTS over $\Sigma = (\Omega, A, C)$. Let us note $\Sigma_F = (\Omega, F, C)$. A* full symbolic execution of G over F *is a triple $(\mathcal{S}, init, R)$ with $init = (q_0, true, \sigma_0)$ where σ_0 is an injective substitution in F^A and $R \subseteq \mathcal{S} \times Act(\Sigma_F) \times \mathcal{S}$ such that for any two transitions in R respectively of the form $(\eta^i, c?x, \eta^f)$ and $(\eta'^i, d?y, \eta'^f)$, the variables x and y are distinct and $\forall a \in A, \sigma_0(a) \neq x$. For any $\eta \in \mathcal{S}$ of the form*

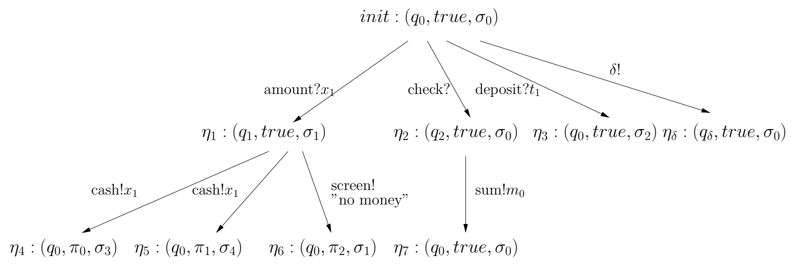
[4] Let us recall that here, π is *satisfiable* if and only if there exists $\nu \in M^F$ such that $M \models_{\nu} \pi$ since variables of π are by construction in F.

(q, π, σ), *for all* $tr \in Trans$ *of the form* $(q, act, \varphi, \rho, q')$, *there exists an unique symbolic transition* $st = (\eta, sa, \eta')$ *in* R *such that*

- *if* $act = c!t$ *(resp.* $c!$*), then* $sa = c!\sigma(t)$ *(resp.* $c!$*) and* $\eta' = (q', \pi \wedge \sigma(\varphi), \sigma \circ \rho)$,
- *if* $act = c?x$ *with* x *in* A *then* $sa = c?z$ *with* z *in* F, *and* $\eta' = (q', \pi \wedge \sigma(\varphi), \sigma \circ (x \mapsto z) \circ \rho)$,
- *if* $act = c?$ *then* $sa = c?$, *and* $\eta' = (q', \pi \wedge \sigma(\varphi), \sigma \circ \rho)$.

The symbolic execution *of* G *over* F *is the triple* $SE(G) = (\mathcal{S}_{sat}, init, R_{sat})$ *where* R_{sat} *is the restriction of* R *to* $\mathcal{S}_{sat} \times Act(\Sigma_F) \times \mathcal{S}_{sat}$.

The trace semantics for a symbolic execution tree is defined in a natural way. If one solves the path condition of a given path (i.e. the path condition of its last state) one can then evaluate all symbolic actions labelling this path and extract the corresponding trace. Since $SE(G)$ is obtained from the symbolic execution tree of G by removing only un-solvable paths, one can easily prove that $Trace(G) = Trace(SE(G))$. Finally, since an IOSTS and its symbolic execution share the same trace semantics, it is equivalent to study an IOSTS or its symbolic execution in the context of conformance testing.



$init : (q_0, true, \sigma_0)$
amount?x_1
check?
deposit?t_1
$\delta!$
$\eta_1 : (q_1, true, \sigma_1)$
$\eta_2 : (q_2, true, \sigma_0)$
$\eta_3 : (q_0, true, \sigma_2)$
$\eta_\delta : (q_\delta, true, \sigma_0)$
cash!x_1
cash!x_1
screen! "no money"
sum!m_0
$\eta_4 : (q_0, \pi_0, \sigma_3)$
$\eta_5 : (q_0, \pi_1, \sigma_4)$
$\eta_6 : (q_0, \pi_2, \sigma_1)$
$\eta_7 : (q_0, true, \sigma_0)$

$\sigma_0 = x \to x_0, m \to m_0, t \to t_0, cp \to cp_0$ $\qquad \pi_0 = cp_0 \leq 2 \wedge x_1 \leq m_0 \wedge m_0 \leq 1000$

$\sigma_1 = x \to x_1, m \to m_0, t \to t_0, cp \to cp_0 + 1$ $\qquad \pi_1 = cp_0 \leq 2 \wedge x_1 \leq m_0 \wedge m_0 < 1000$

$\sigma_2 = x \to x_0, m \to m_0 + t_1, t \to t_1, cp \to cp_1$ $\qquad \pi_2 = cp_0 > 2 \vee x_1 > m_0$

$\sigma_3 = x \to x_1, m \to m_0 - x_1, t \to t_0, cp \to cp_0 + 1$

$\sigma_4 = x \to x_1, m \to m_0 - x_1 - 1, t \to t_0, cp \to cp_0 + 1$

Fig. 2. Symbolic execution tree

Example 2. Figure 2 illustrates the beginning of the symbolic execution of the ATM system presented in Figure 1.

3.2 Inclusion criterion

A reactive system is supposed to continuously interact with its environment. Thus, behaviours viewed as sequences of interactions are very often arbitrary long. It explains that IOSTS specifications of reactive systems often contain internal loops. This implies that the symbolic execution of the corresponding

IOSTS has infinite paths. However, in practice, one can consider an arbitrary long behaviour as a sequence of "basic" behaviours. For example, the ATM system basically offers few basic behaviours. It may: (1) provide the user with money, (2) receive deposit from the user or (3) give the current level of the user account. Any "complex" behaviour of the ATM system can be seen as a sequence of such basic behaviours. Now if one considers the symbolic execution of the ATM system, one would observe a lot (or even an infinite number) of occurrences of those basic behaviours. In other words, information on symbolic behaviours provided by the symbolic execution may be highly redundant in terms of basic behaviours. We propose to cut the symbolic execution of an IOSTS in order to lower this redundancy. Definition 9 of symbolic execution shows that behaviours are indeed determined by states, that is why our procedure to cut the tree is grounded on a relation upon states. From a symbolic state $\eta = (q, \pi, \sigma)$ one can extract constraints on the set A of attribute variables : the set of all possible interpretations $\nu_A : A \to M$ corresponding to η are restrictions[5] to A of all interpretations $\nu : A \cup F \to M$ such that[6] $M \models_\nu \bigwedge_{x \in A}(x = \sigma(x)) \wedge \pi$. If the set of possible interpretations of A for η_1 is included in the one of η_2 one says that $\eta_1 \subseteq \eta_2$.

Definition 10 (States inclusion). *Let $\eta = (q, \pi, \sigma)$ and $\eta' = (q, \pi', \sigma')$ be two symbolic extended states with resp. F_η and $F_{\eta'}$ as subsets of variables in F occurring resp. in π and π'. $\eta \subseteq \eta'$ iff, if for any $\nu : A \cup F_\eta \to M$ such that $M \models_\nu (\wedge_{x \in A}(x = \sigma(x)) \wedge \pi)$ then there exists $\nu' : A \cup F_{\eta'} \to M$ such that $\nu_{|A} = \nu'_{|A}$ and $M \models_{\nu'} (\wedge_{x \in A}(x = \sigma'(x)) \wedge \pi')$.*

Let us consider η_1 and η_2 verifying $\eta_1 \subseteq \eta_2$. Any transition that can be symbolically executed from η_1 can also be symbolically executed from η_2. Moreover if one executes a transition t from η_1 and from η_2, this results in two target symbolic extended states η_1' and η_2' such that $\eta_1' \subseteq \eta_2'$. Recursively applying this reasoning step allows one to deduce that any symbolic behaviour that can be deduced from η_1 can also be deduced from η_2. Then we propose to consider a reduced symbolic execution by removing the subtree of root η_1.

Definition 11 (Restriction by inclusion). *Let $SE(G) = (\mathcal{S}_{sat}, init, R_{sat})$ be a symbolic execution of G. A* restriction of $SE(G)$ satisfying the inclusion criterion *is a triple $SE(G)^{\subseteq} = ({\mathcal{S}_{sat}}^{\subseteq}, init, {R_{sat}}^{\subseteq})$ where:*

- *${\mathcal{S}_{sat}}^{\subseteq} \subseteq \mathcal{S}_{sat}$, $init \in {\mathcal{S}_{sat}}^{\subseteq}$, and ${R_{sat}}^{\subseteq} \subseteq R_{sat}$.*
- *For any $\eta \in {\mathcal{S}_{sat}}^{\subseteq}$ if there is no $(\eta, sa, \eta') \in {\mathcal{R}_{sat}}^{\subseteq}$ then either there is no $(\eta, sa, \eta') \in \mathcal{R}_{sat}$ or there exists $\eta'' \in {\mathcal{S}_{sat}}^{\subseteq}$ such that $\eta \subseteq \eta''$.*
- *For any $\eta \in {\mathcal{S}_{sat}}^{\subseteq}$, if there exists $(\eta, sa, \eta') \in {\mathcal{R}_{sat}}^{\subseteq}$ then for all $(\eta, sa', \eta'') \in \mathcal{R}_{sat}$, $(\eta, sa', \eta'') \in {\mathcal{R}_{sat}}^{\subseteq}$.*

[5] As usual, the restriction of an application $f : X \to Y$ to a subset Z of X will be denoted by $f_{|Z}$.

[6] When reading $x = \sigma(x)$ for $x \in A$ in the formula, the reader should be aware that $\sigma(x)$ denotes in fact an expression in terms of variables of F.

Definition 11 does not require that the restriction gets a finite number of symbolic extended states: it may happen that symbolic extended states cannot be compared through $\subseteq$. However, in practice, reactive systems generally have the property that they regularly come back to already encountered states, as for example the initial state. For such systems, the restriction by inclusion of their symbolic execution generally gives a finite tree.

Example 3. Figure 2 corresponds in fact to a restriction by inclusion of the symbolic execution of the ATM system. Indeed, $\eta_4 \subseteq init$ since η_4 contains the same state q_0 as *init* and the constraints in η_4, i.e. $\pi_0 = cp_0 \leq 2 \wedge x_1 \leq m_0 \wedge m_0 \geq 1000$, are stronger that those in *init* (*true*). The symbolic extended states η_3, η_5, η_6 and η_7 are handled in the same way.

4 Conformance testing for IOSTS

4.1 Our approach

Conformance testing supposes that a formal conformance relation is given between the specification G and the system under test SUT. We propose to adapt the *ioco* relation used for example in [12]. As usual for conformance testing, we consider that the SUT is only observable by its input/output sequences. In particular, data handled in these sequences are concrete values which may be denoted by ground terms of T_{Ω}. By hypothesis, the SUT may be modelled as a labelled transition system for which transitions are simple emissions (output) or receptions (input) carrying concrete values. Moreover, as usual, the SUT is supposed to accept all inputs in all states. The set of traces which can be observed for the SUT, denoted by $Trace(SUT)$, is a subset[7] of $(Act(M) \cup \{(\delta, !)\})^*$. Intuitively a SUT is conform to its specification with respect to *ioco* if the reactions of the SUT are the same than those specified when it is stimulated by inputs deduced from the specification.

Definition 12. *SUT conforms to G if and only if for any $tra \in Trace(G_\delta) \cap Trace(SUT)$, if there exists $act \in Act(M) \cup \{(\delta, !)\}$ of the form $(c, !, t)$ or $(c, !)$ such that $tra.act \in Trace(SUT_\delta)$, then $tra.act \in Trace(G_\delta)$.*

Test purposes are used to select some behaviours to be tested. In our case, test purposes consist of some finite paths of the symbolic execution of the specification. For each of those paths, the last symbolic extended state is the target state of an output action and is labelled by the keyword *accept*. All states belonging to a chosen path (except the last one labelled by *accept*) are labelled by *skip*. So, a *skip* label simply means that it is still possible to reach an *accept* state by emitting or receiving additional messages . So, a test purpose is a finite subtree of the symbolic execution whose leaves are labelled by *accept* and intermediate nodes are labelled by *skip*. All other states, external to the test

[7] The absence of outputs from SUT can be observed through the emission $\delta!$, and in this case, this cannot be directly followed by another emission.

purpose, are labelled by $\odot$: they are not meaningful with respect to the selected paths of the test purpose.

Definition 13. *Let G be an IOSTS with $SE(G_\delta) = (\mathcal{S}_{sat}, init, R_{sat})$ its associated symbolic execution. A* symbolic test purpose *for G is an application $TP : \mathcal{S}_{sat} \to \{skip, accept, \odot\}$ such that:*

- *there exists η verifying $TP(\eta) = accept$,*
- *for any η, η' verifying $TP(\eta) = TP(\eta') = accept$, there is no finite path $st_1 \cdots st_n$ such that for some $i \leq n$, $source(st_i) = \eta$ and $target(st_n) = \eta'$,*
- *for any η' verifying $TP(\eta') = accept$, there exists (η, sa, η') in $SE(G_\delta)$ such that sa is of the form $c!t$ or $c!$.*
- *$TP(\eta) = skip$ iff there exists a finite path $st_1 \cdots st_n$ such that for some $i \leq n$, $source(st_i) = \eta$ and $TP(target(st_n)) = accept$. Otherwise $TP(\eta) = \odot$.*

Unlike [5], our test purposes directly characterize by construction a subset of the specified behaviours since they are extracted from the symbolic execution of the specification. In the following sections, the considered test purposes will refer to an arbitrary test purpose generically denoted by TP.

4.2 Preliminary definitions and informal description

A test execution consists in executing on the SUT a transition system, called a test case and devoted to produce testing verdicts as $PASS$ or $FAIL$. The test case and the SUT share the same set of channels and are synchronized by coupling emissions and receptions on a given communication channel. We focus on the sequence of data exchanged between the test case and the SUT. These data are in fact elements of M (the model of the data part) and will be denoted by ground terms of T_Ω. We use the following notations: $obs(c!t)$ with t in T_Ω to characterize that the SUT emits through the channel c the concrete value denoted t and $stim(c?t)$ to represent stimulations of the SUT, occurring when the data t is sent by the test case to the SUT. We also use the following generic notation $[ev_1, ev_2, \ldots, ev_n | Verdict]$ for a sequence of synchronized transitions between a test case and the SUT leading to the verdict $Verdict$, each action ev_i being issued either from an observation $obs(ev_i)$ or a stimulation $stim(ev_i)$.

Testing a SUT with respect to a given symbolic test purpose amounts to look for stimulating and observing the SUT in such a way that when conformity is not violated, the sequence of stimulations and observations corresponds to a trace (belonging to semantics) of at least one path of the test purpose.

To reach this goal, the testing process achieves two tasks. The first one consists in computing, each time it is required, a stimulation compatible with reaching an *accept* state. The second one consists in computing all the symbolic states which may have been reached taking into account the whole sequence of observations/stimulations already encountered.

We firstly define *contexts* composed of a symbolic state and of a formula expressing constraints induced by the sequence of previously encountered inputs/outputs.

Definition 14 (Context). *A context is a couple (s, f) where $s \in \mathcal{S}_{sat}$ and f is a formula whose variables are in F.*

As previously pointed out, there may be more than one single context compatible with a sequence of observations/stimulations. This is taken into account by using a set of contexts, generically noted SC (for Set of Contexts), representing the set of all potential appropriate contexts for a given sequence of stimulations/observations. We introduce some auxiliary functions useful to reason about sets of contexts, in particular in order to be able to compute the sequence of sets of contexts resulting from the successive application of elementary actions.

Definition 15 (Function $Next(ev, SC)$). *Let SC be a finite set of contexts and $ev \in Act(\Sigma_F)$. If ev is of the form $c\triangle t$ (resp. $c\triangle$) with $\triangle \in \{?, !\}$ then $(s', f') \in Next(ev, SC)$ with $s' = (q', \pi', \sigma')$ iff:*

- *there exists $(s, f) \in SC$ such that $(s, c\triangle u, s') \in R$ (resp. $(s, c\triangle, s') \in R$)*
- *f' is $f \wedge (t = u)$ (resp. f) and $f' \wedge \pi'$ is satisfiable.*

Thus, $Next(ev, SC)$ computes the set of all contexts following directly the context SC with the event ev. When stimulating the SUT, it matters to check whether the computation of a stimulation is compatible with the goal of finally reaching an *accept* state. For that, for any context *ct*, the $targetCond(ct)$ predicate allows us to confront constraints inherited from the first observations or stimulations to the target states, those labelled by *accept* by the test purpose.

Definition 16 ($targetCond(ct)$). *Let $ct = (s, f)$ be a context such that $TP(s) = skip$ and[8] $E = \{s' \in \mathcal{S}_{sat} \mid \exists m \in (Act(\Sigma_F))^*,\ s \xrightarrow{m} s'$ and $TP(s') = accept\}$, then $targetCond(ct)$ is the formula : $\bigvee_{(q,\pi,\sigma) \in E} \pi$.*

Given a set of contexts SC, we distinguish among all contexts in $Next(ev, SC)$ those which are pertinent with respect to the considered test purpose:

Definition 17 (Functions $NextSkip(ev, SC)$ and $NextPass(ev, SC)$). *Let SC be a finite set of contexts and $ev \in Act(\Sigma_F)$. If ev is of the form $c\triangle t$ (resp. $c\triangle$) with $\triangle \in \{?, !\}$ then $(s', f') \in NextSkip(ev, SC)$ iff:*
- *there exists $(s, f) \in SC$ such that $(s, c\triangle u, s') \in R$ (resp. $(s, c\triangle, s') \in R$) with $TP(s') = skip$*
- *f' is $f \wedge (t = u)$ (resp. f) and $f' \wedge targetCond(s')$ is satisfiable.*

$NextPass(ev, SC)$ is defined in the same way with the difference that $TP(s')$ is required to be accept instead of skip.

Let us remark that for a given symbolic state $s' = (q', \pi', \sigma')$, the predicate $targetCond(s')$ is necessarily stronger[9] than π' since by definition of symbolic

[8] For a labelled graph G and a word $m = a_1 \cdot \cdots . a_n$, the notation $s_0 \xrightarrow{m} s_n$ stands for any path $s_0 \xrightarrow{a_1} s_1 \cdots s_{n-1} \xrightarrow{a_n} s_n$ where each $s_i \xrightarrow{a_i} s_{i+1}$ is a transition of G.

[9] π' is said to be stronger than π iff for any interpretation ν, if $M \models_\nu \pi'$, then $M \models_\nu \pi$.

execution, the set of constraints is increasing at each new transition. Thus, we get $NextSkip(ev, SC) \subseteq Next(ev, SC)$ and $NextPass(ev, SC) \subseteq Next(ev, SC)$ for all contexts SC and events ev. Emptiness of $NextSkip(ev, SC)$ means that no more *accept* is now reachable while non emptiness of $NextPass(ev, SC)$ means that at least an *accept* has been reached.

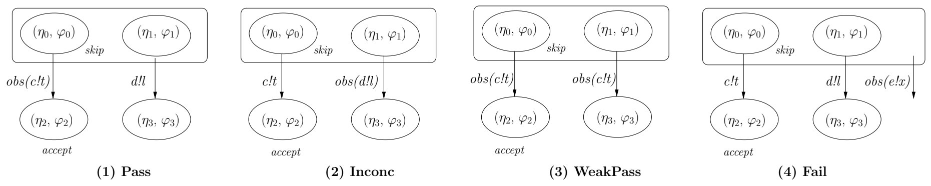


Fig. 3. Algorithm's explanations

Let us illustrate our algorithm with Figure 3 and describe an execution step based on an emission ev from the SUT and starting from $SC = \{(\eta_0, \varphi_0), (\eta_1, \varphi_1)\}$. If $Next(ev, SC)$ is empty, that is the case for $ev = e!x$, this means that the emission is not specified and so we conclude $FAIL$ (see Figure 3 (4)). If an *accept* is reached ($NextPass(ev, SC)$ non empty) we conclude $PASS$ when no other context is reached, see for example Figure 3 (1) with $NextPass(c!t, SC) = \{(\eta_2, \varphi_2)\}$, or $WeakPASS$ when others contexts are also reached, see for example Figure 3 (3) with $Next(c!t, SC) = \{(\eta_2, \varphi_2), (\eta_3, \varphi_3)\}$. In this last case, we cannot distinguish whether the inner state of the SUT is represented by the reached *accept* state (η_2, φ_2) or by the state (η_3, φ_3) outside of the test purpose. At last, if $NextSkip(ev, SC)$ is empty while $Next(ev, SC)$ is not, see Figure 3 (2) for $ev = d!l$, this means that the emission was specified but was not aimed by the test purpose. Then, we conclude by an inconclusive verdict $INCONC$.

4.3 Inference rules

Let us recall that our goal is to compute sequences $[ev_1, \dots, ev_n | Verdict]$ representing synchronized transitions between a test case and the SUT leading to the verdict $Verdict$, each action ev_i being derived either from an observation $obs(ev_i)$ or a stimulation $stim(ev_i)$, and $Verdict$ belonging to this set of keywords : $\{PASS$, $WeakPASS$, $INCONC$, $FAIL\}$. For that, we will take into account the knowledge of the associated contexts. Each step of the construction of such a sequence will be described by means of inference rules. Those rules are structured as follows[10] $\frac{SC}{Result}$ $cond(ev)$ where SC is a set of contexts, $Result$ is either a set of contexts or a verdict, $cond(ev)$ is a set of conditions including the observation $obs(ev)$ or the stimulation $stim(ev)$. One should read a rule as follows: *Given the current set of contexts SC, if* $cond(ev)$ *is verified then the algorithm may achieve a step of execution, with ev as elementary action.* As long as

[10] The initialisation rule will not respect this generic structure since it will simply consist in introducing the starting context.

Result is a set of contexts, a new rule may be applied to pursue the computation of the sequence. And of course, reaching a verdict stops the algorithm.
Rule 0: Initialisation rule

$$\frac{}{\{(init, true)\}}$$

Rule 1: The emission is compatible with the purpose but no *accept* is reached.

$$\frac{SC}{Next(ev, SC)} \quad obs(ev), NextSkip\ (ev, SC) \neq \emptyset, NextPass(ev, SC) = \emptyset$$

Rule 2: The emission is not expected with regards to the specification.

$$\frac{SC}{FAIL} \quad obs(ev), Next(ev, SC) = \emptyset$$

Rule 3: The emission is specified but not compatible with the test purpose.

$$\frac{SC}{INCONC} \quad obs(ev), Next(ev, SC) \neq \emptyset, NextSkip(ev, SC) = \emptyset, NextPass(ev, SC) = \emptyset$$

Rule 4: All next contexts are *accept* ones.

$$\frac{SC}{PASS} \quad obs(ev), Next(ev, SC) = NextPass(ev, SC), Next(ev, SC) \neq \emptyset$$

Rule 5: Some of the next contexts are labelled by *accept*, but not all of them.

$$\frac{SC}{WeakPASS} \quad obs(ev), NextPass(ev, SC) \neq \emptyset, NextPass(ev, SC) \subsetneq Next(ev, SC)$$

Rule 6: Stimulation of the *SUT*

$$\frac{SC}{Next(ev, SC)} \quad stim(ev), NextSkip\ (ev, SC) \neq \emptyset$$

Rules from 1 to 5 concern observations while only Rule 6 concerns stimulations. Rule 5 calls for some comments: a verdict $WeakPASS$ means both that the test purpose is reached and that the sequence of observations/stimulations may correspond to another behaviour of the symbolic execution. This verdict is thus a kind of *warning*. One should pursue the test execution sequence to distinguish which states really correspond to the performed execution sequence.

We can consider a transition system, denoted $TS(TP)$, from a test purpose TP and the set of inference rules. The states are the sets of contexts appearing in the rules and four special states labelled by the verdicts. Two states are related by a transition labelled by an emission $ev = c!t$ or $ev = c!$ (resp. a receipt $ev = c?t$ or $ev = c?$) if they can be relied by the application of the unique rule conditioned by $stim(ev)$ (resp. of one of the rules conditioned by $obs(ev)$). Such a transition system is a simple labelled one. If such a transition system is synchronized with the system under test in such a way that emissions and receptions are synchronized by sharing the same communication channel and the same data, then any licit sequence of synchronized transitions is necessarily finite and leads to one of the four verdicts. In fact, this transition system may be viewed as a test case in the sense of [4], except that our transition system may be non-deterministic. Indeed, for a given set of contexts, several rules may be applied. In particular, depending of the form of the specification, one can choose to send to the system a message, to wait for an emission or to observe quiescence. Even worse, for a given rule, several choices are often possible for the data carried by the associated observation or stimulation.

We note $st(TP, SUT)$ the set of $[ev_1, \ldots, ev_n|Verdict]$ such that $ev_1 \ldots ev_n$ is a sequence of synchronized transitions between $TS(TP)$ and SUT leading to the final state labelled by $Verdict$ in $TS(TP)$. Finally, we introduce the notation: $vdt(TP, SUT) = \{Verdict \mid \exists ev_1, \ldots ev_n, [ev1, \ldots, evn|Verdict] \in st(TP, SUT)\}$

Using these notations, we can now state the correctness and the completeness of our algorithm:

Theorem 1. *For any IOSTS G and any SUT:*

Correctness: *If SUT conforms to G, for any symbolic test purpose TP, $FAIL \notin vdt(TP, SUT)$.*

Completeness: *If SUT does not conform to G, there exists a symbolic test purpose TP such that $FAIL \in vdt(TP, SUT)$.*

The completeness property holds up to all the non-deterministic choices induced by our set of rules and captured in the set $vdt(TP, SUT)$.

5 Criterion-based test purposes

Most of the times, the set of all finite symbolic behaviours associated to a specification is lucky enough to be infinite. In such a case, one generally uses coverage criteria to define test purposes.

The first idea is to simply cut the (infinite) symbolic execution of a specification according to a parameter n indicating the length of the paths to be tested. The corresponding test purpose will contain all the paths of length n derived from the symbolic execution, provided that they are terminated by an output action.

Definition 18 ("all paths of length n"). *Let G be an IOSTS on the signature Σ and let us consider $SE(G_\delta) = (\mathcal{S}_{sat}, init, R_{sat})$ its associated symbolic execution. Let $n \geq 0$. The* test purpose "all paths of length n" *for G is the test purpose $TG_n : \mathcal{S}_{sat} \rightarrow \{skip, accept, \odot\}$ such that the only symbolic states labelled by accept by TG_n are given by the following property. For any path $p = t_1 \cdots t_n$ of $SE(G_\delta)$ starting from init and verifying $length(p) = n$:*

- *either $act(t_n) \in Output(\Sigma_\delta)$ and $TG_n(target(t_n)) = accept$,*
- *or for any*[11] *$p.t \in Ext_{out}(p)$, $label(target(t)) = accept$.*

The criterion "all paths of length n" allows one to characterize a countable family of test purposes, approaching more and more the whole symbolic execution of the specification. Then, the tester can make an trade-off between the size of the test purpose (in relation with the parameter n) and the testing cost.

[11] Let us recall that by extension, symbolic executions of IOSTS inherit from notions associated to IOSTS: here, we have translated the notion Ext_{out} defined for IOSTS.

Moreover, such a test purpose may be decomposed in as many test purposes as *accept* states: indeed, for each *accept* state, we can build a dedicated test purpose with this state as unique *accept* state. Such a decomposition allows the tester to systematically try to reach each *accept* state, thus, to reach each path of length n (up to the fact that they are not necessarily terminated by an output action).

The criterion "all paths of length n" allows us to build test purposes. However, the pertinent length n to be chosen is up to the tester. In order to help the tester to chose this parameter n, we propose to use the restriction by inclusion defined in Definition 11. This characterizes a subpart of a symbolic execution with no redundant behaviours. The inclusion criterion gives some clear indications about the size of basic behaviours of the specification. Intuitively, one can choose for the value of the parameter n the length p_{max} of the longest path of a restriction by inclusion of a symbolic execution. More generally, one can compose basic behaviours by juxtaposing them. It suffices to take for the parameter n, p_{max}, $2 \times p_{max}, \ldots$ or $k \times p_{max}$ if we want to consider all the combinations of k basic behaviours.

Definition 19 ("k-inclusion" criterion).
Let G be an IOSTS and $SE(G_\delta)^{\subseteq} = ({\mathcal{S}_{sat}}^{\subseteq}, init, R^{\subseteq})$ be a restriction of $SE(G_\delta)$ satisfying the inclusion criterion. Let us note $p_{max} \in FP(SE(G_\delta)^{\subseteq})$ such that for all $p \in FP(SE(G_\delta)^{\subseteq})$, $length(p_{max}) \geq length(p)$. Let $k > 0$. The test purpose "k-inclusion criterion" *associated to $SE(G_\delta)^{\subseteq}$ is the test purpose "all paths of length $k \times length(p_{max})$" for G.*

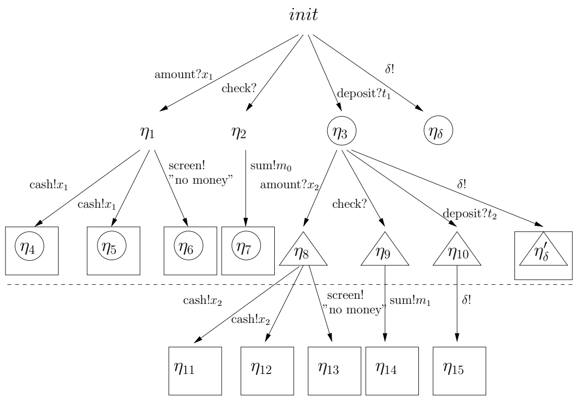


Fig. 4. Construction of test purposes

Example 4. Figure 4 illustrates the construction of test purposes. The leaves of the symbolic tree constructed in Figure 2, *i.e.* the restriction by inclusion of the ATM system, are represented by a circle $\bigcirc$. This tree is completed from the symbolic state η_3 with symbolic states (that are represented by a triangle $\triangle$) to have the same length for all paths of the tree (the dotted line marks the length 2). For each leaf above the dotted line which does not result from an output, some additional outputs are considered to ensure that paths to be tested are observed by outputs. This last step introduces the states η_{11}, η_{12}, η_{13}, η_{14}, η_{15}. Finally, the states that are in a square $\square$ are those labelled with *accept*. It corresponds to the 1-inclusion criterion (for lack of space, we cannot unfold the symbolic tree until the 2-inclusion criterion but it would be the same construction). Now, we can apply the rules of our algorithm over the paths of the symbolic tree of Figure 4. We explain the computation of the final verdict by making explicit the intermediate applications of rules over the current set of contexts (SC) using the following notation : $SC \xrightarrow[rule]{action} SC'$ where *action* denotes the current element of the considered trace (either of the form $c!t$, $c!$, $c?t$ or $c?$), *rule* indicates which rule is applied to get the next set of context SC'.

Let us consider the trace $[deposit?250\ amount?50\ cash!50 \mid WeakPASS]$.

$$SC_0 = \{(init, true)\} \xrightarrow[rule6]{deposit?250} SC_1 = \{(\eta_3, t_1 = 250)\} \xrightarrow[rule6]{amount?50} SC_2$$

$$SC_2 = \{(\eta_8, (t_1 = 250 \wedge x_2 = 50))\} \xrightarrow[rule5]{cash!50} WeakPASS$$

The $WeakPASS$ verdict is due to the 2 equalities :
- $Next(cash!50, SC_2) = \{(\eta_{11}, (t_1 = 250 \wedge x_2 = 50)), (\eta_{12}, (t_1 = 250 \wedge x_2 = 50))\}$
- $NextPass(cash!50, SC_2) = \{(\eta_{11}, (t_1 = 250 \wedge x_2 = 50))\}$.

One cannot decide whether the test purpose has been achieved (the real state corresponds to η_{11}) or missed (the real state corresponds to η_{12}).

5.1 Implementation issues

The work presented here is implemented as an extension of the $AGATHA$ tool set [7,10] which uses symbolic execution techniques to debug and validate specifications. The $AGATHA$ tool allows to unfold IOSTS specifications in the form of trees provided with path conditions for all paths of trees. Trees are computed according to coverage criteria including those grounding test purpose definitions discussed in Section 5. Those test purposes are thus obtained for free. All rules defined in Section 4.3 are implemented. However applying those rules does not necessarily lead to a deterministic process. Implementing deterministic strategies for rules appliance is still an open issue. Presburger arithmetics [8] constitutes the data part of IOSTS treated by AGATHA. The algorithm requires some decision procedures (for inclusion criterion) and constraint solving (to compute stimulations). This is done thanks to the Omega Library [1].

6 Conclusion

We have proposed an approach to test reactive systems specified as Input/ Output Symbolic Transition Systems (IOSTS). Symbolic execution allows us to re-express the specification in the form of a tree whose set of paths denotes the set of all behaviours of the specification. We propose to define test purposes by selecting a finite set of behaviours (*i.e.* paths) in the symbolic execution tree.

We define an algorithm to test SUT with regard to a test purpose. This algorithm is given by a set of rules, both to compute stimulations of SUT which are adequate to achieve the test purpose and to assign a verdict to a test execution. There may be four verdicts: $PASS$, $FAIL$, $INCONC$ and $WeakPASS$. $WeakPASS$ is a verdict which expresses that conformance is observed but we are not able to ensure that the test purpose is really achieved. Indeed, it may happen that an input/output sequence observed during a test execution can be related to several behaviours, not all being accepting paths of the test purpose.

Test purposes may be defined manually but we also propose to use some coverage criteria to automatically extract them. The first one is the *all symbolic behaviours of length* n criterion which requires to cover all paths of length n in the symbolic execution. The other one, so-called *restriction by inclusion* criterion, extracts a subset of all paths of the symbolic execution tree by avoiding redundancies. Concerning coverage criteria, we are currently investigating other kinds of criteria. Our aim is to help the tester in defining test purposes. Indeed on the one hand test purposes are difficult to define manually as soon as the specification has a realistic size, but on the other hand the intervention of a human is often necessary to characterize "clever" test purposes (*i.e.* allowing to discover subtle non conformance).

References

1. Omega 1.2. The Omega Project: Algorithms and Frameworks for Analyzing and Transforming Scientific Programs. 1994.
2. L.-A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on software engineering*, 2(3):215–222, September 1976.
3. L. Frantzen, J. Tretmans, and T. A.C. Willemse. Test generation based on symbolic specifications. In J. Grabowski and B. Nielsen, editors, *FATES 2004*, number 3395 in LNCS, pages 1–15. Springer-Verlag, 2005.
4. C. Jard and T. Jéron. TGV: theory, principles and algorithms, a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Software Tools for Technology Transfer (STTT)*, 6, October 2004.
5. 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 the Construction and Analysis of Systems (TACAS)*, volume 3440, Edinburgh, April 2005.
6. J.-C. King. A new approach to program testing. *Proceedings of the international conference on Reliable software, Los Angeles, California*, 21-23:228–233, April 1975.
7. D. Lugato, N. Rapin, and J.-P. Gallois. Verification and tests generation for SDL industrial specifications with the AGATHA toolset. In P. Petterson and S. Yovine, editors, *Proceedings of the Workshop on Real-Time Tools affiliated to CONCUR01*, Department of Information Technology UPPSALA UNIVERSITY Box 337, SE-751 05 Sweden, August 2001. ISSN 1404-3203.
8. M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetic. *Comptes rendus du premier Congres des Math. des Pays Slaves*, pages 92–101,395, 1929.

9. C.-V. Ramamoorthy, S.-F. Ho, and W.-T. Chen. On the automated generation of program test data. *IEEE Transactions on software engineering*, 2(4):293–300, September 1976.
10. N. Rapin, C. Gaston, A. Lapitre, and J.-P. Gallois. Behavioural unfolding of formal specifications based on communicating automata. In *Proceedings of first Workshop on Automated technology for verification and analysis*, Taiwan, 2003.
11. 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.
12. J. Tretmans. Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.
13. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.