A Logic for Assessing Sets of Heterogeneous Testing Hypotheses *

Ismael Rodríguez, Mercedes G. Merayo and Manuel Núñez

Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, 28040 Madrid, Spain
isrodrig@sip.ucm.es,mgmerayo@fdi.ucm.es,mn@sip.ucm.es

Abstract. To ensure the conformance of an *implementation under test* (IUT) with respect to a specification requires, in general, the application of an infinite number of tests. In order to use finite test suites, most testing methodologies add some feasible hypotheses about the behavior of the IUT. Since these methodologies are designed for considering a *fix* set of hypotheses, they usually do not have the capability of dealing with other scenarios where the set of assumed hypotheses varies. We propose a logic to infer whether a set of *observations* (i.e., results of test applications) allows to claim that the IUT conforms to the specification *if* a specific set of hypotheses (taken from a repertory) is assumed.

1 Introduction

The time a tester can spend testing an IUT with respect to a specification is finite, whereas IUTs define, in general, arbitrarily long behaviors. Hence, it takes infinite time to assess the validity of all these behaviors with respect to a specification. In order to overcome this problem, testers add some reasonable assumptions about the implementation regarding the knowledge about its construction. For example, the tester can suppose that the implementation can be represented by means of a deterministic finite state machine, that it has at most n states, etc. A lot of testing methodologies have been proposed which, for a specific set of initial hypotheses, guarantee that a test suite extracted from the specification is correct and complete to check the conformance of the IUT with respect to the specification (e.g. [2, 8, 15, 12]).

However, a framework of hypotheses established in advance is very strict and limits the applicability of a specific testing methodology. For example, in a concrete environment, the tester could assume that the behavior in four specific states of the implementation is deterministic and that two of them represent equivalent states of the implementation. The tester could also make more complex assumptions such as "*non-deterministic states of the IUT cannot show outputs that the machine did not show once the state has been tested 100 times.*"

* Research partially supported by the Spanish MCYT project TIC2003-07848-C02-01, the Junta de Castilla-La Mancha project PAC-03-001, and the Marie Curie project MRTN-CT-2003-505121/TAROT.

In a different scenario the tester could not believe this but think that "*if she observes two sequences of length 200 and all their inputs and outputs coincide then they actually traverse the same IUT states.*" If the tester assumes the validity of a set of hypotheses to test a given IUT, then a specific test suite would be appropriate, while by using other hypotheses, the test suite could not be so.

It would be desirable to provide the tester with a tool to let her analyze the impact of considering a given set of hypotheses in the testing process, as well as the consequences of adding/eliminating hypotheses from the set. The goal of this methodology would be to ascertain if a given finite set of observations extracted by a test suite is *complete* in the case that the considered hypotheses hold, that is, we assess whether obtaining these observations from the IUT implies that the IUT conforms to the specification *if* the hypotheses hold. In this paper we propose a *logic* called $\mathcal{HOTL}$ (*Hypotheses and Observations Testing Logic*). Its aim is to assess whether a given set of observations implies the correctness of the IUT under the assumption of a given set of hypotheses. In order to allow the tester to compose sets of hypotheses, the logic provides a repertory of hypotheses, including some of the ones appearing in known testing methodologies.

Our logic allows to perform at least *three* different tasks. First, a tester can use it to customize the testing process to her specific environment. By using the logic, she can infer not only the consequences of adding a new test, but also the consequences of adding a new hypothesis. In this way, the tester has control over a wide range of testing variables. In particular, the construction of test suites to extract observations and the definition of hypotheses can influence each other. This provides a dynamic testing scenario where, depending on the specification and the tester's knowledge of the IUT, different sets of tests and hypotheses can be considered. Second, such logic allows the tester to evaluate the *quality* of a test suite to discover errors in an implementation: If the observations that could be extracted by the test suite require (for their completeness) a set of hypotheses that is *harder* to be accepted than those required by another suite, then the latter suite should be preferred. This is because this suite could allow the tester to reach diagnostics in a less restrictive environment. Finally, our logic provides a conceptual bridge between different testing approaches. In particular, we may use it to represent the (fix) sets of hypotheses considered by different approaches. Then, by considering the observations each test suite could obtain, a test suite that is complete in an approach could be turned into a complete suite in another. Similarly, we can analyze how the size of test suites is affected by hypotheses. Moreover, we can use the logic to create intermediate approaches where sets of hypotheses are appropriately mixed.

Let us concentrate on how our logic is applied to perform the first of the previous tasks, that is, serving as core of a (dynamic) testing methodology. The methodology is applied in two phases. The first phase consists in the classical application of tests to the IUT. By using any of the available methods in the literature, a test suite will be derived from the specification. If the application of this test suite finds an unexpected result then the testing process stops: The IUT is not conforming. However, if such a wrong behavior is not detected then

the tester cannot be sure that the IUT is correct. In this case, the second phase begins, that is, the tester applies the logic described in this paper to infer whether passing these tests *implies* that the IUT is correct if a given set of hypotheses is assumed. If it does then the IUT is assumed to be correct; otherwise, the tester may be interested in either applying more tests or in assuming more hypotheses (in the latter case, on the cost of *feasibility*) and then applying the logic again until the correctness of the IUT is effectively granted. In order to appropriately apply the logic, the behavior of the IUT observed during the application of tests must be properly represented. For each application of a test to the IUT, we construct an *observation*, that is, a sequence of inputs and outputs denoting the test and the response produced by the IUT, respectively. Both observations and the assumed hypotheses will be represented by appropriate *predicates* of the logic. Then, the deduction rules of the logic will allow to infer whether we can claim that the IUT conforms to the specification (actually, the logic will check whether *all* the implementations that could produce these observations and fulfill the requirements of the hypotheses conform to the specification).

We distinguish two kinds of hypotheses in the predefined repertory: Hypotheses concerning specific parts (states) of the IUT and hypotheses concerning the whole IUT. In order to unambiguously denote the states regarded by the former, they will be attached to the corresponding observations that reached these states. For example, if the IUT was showing the sequence of outputs $o_1, o_2, \ldots, o_n$ as answer to the sequence of inputs $i_1, i_2, \ldots, i_n$ provided by the tester, the tester may think that the state reached after performing i_1/o_1 is deterministic or that the state reached after performing the sequence $i_1/o_1, i_2/o_2$ is the same as the one reached after performing the whole sequence $i_1/o_1, i_2/o_2, \ldots, i_n/o_n$. Let us remark that these are *hypotheses* that the tester is assuming. Thus, she might be wrong and reach a wrong conclusion. However, this is similar to the case when the tester assumes that the implementation is deterministic or that it has at most n states and, in reality, this is not the case. In addition to using hypotheses associated to observations, the tester can also consider global hypotheses that concern the whole IUT. These are assumptions such as the ones that we mentioned before: Assuming that the IUT is deterministic, that is has at most n states, that is has a unique initial state, etc. In order to denote the assumption of this kind of hypotheses, specific logic predicates will be used.

Regarding related work, there are several papers where testing hypotheses are used to perform the testing process. For example, we may consider that the implementation is deterministic (e.g. [13]), that we are testing the coupling of several components by assuming that all of them are correct or that at most one of them is incorrect (e.g. [9]), etc. Our methodology provides a *generalization* of these frameworks because it allows to decide the specific hypotheses we will consider. In this line, we can compare the suitability of different test suites or test criteria in terms of the hypotheses that are considered (e.g. [10]); some formal relations to compare them have been defined [6]. Since our logic provides a mechanism to effectively compare sets of hypotheses, it may help to compute relations defined in these terms. Even though we work with rules and properties,

our work is not related to *model checking* [4] since we do not *check* the validity of properties: We assume that they hold and we infer results about the conformity of the IUT by using this assumption. In the same way, this work is not related to some recent work on passive testing where the validity of a set of properties (expressed by means of *invariants*) is checked by passively observing the execution of the system (e.g. [7, 3, 1]).

The rest of the paper is organized as follows. In Section 2 we present some basic concepts related to the formalisms that we will use. In Section 3 we introduce the predicates of $\mathcal{HOTL}$, while in Section 4 we present the deduction rules. Finally, in Section 5 we present our conclusions and some directions for further research. Due to the lack of space, some auxiliary definitions and rules have not been included in this paper. All of them can be found in [14].

2 Formal Model

In this section we introduce some basic concepts that will be used along the paper to formally present our methodology. Specifically, we introduce the notion of finite state machine and a conformance relation.

Definition 1. A *finite state machine*, in short `FSM`, is a tuple of five elements $M = (\mathcal{S}, \texttt{inputs}, \texttt{outputs}, \mathcal{I}, \mathcal{T})$ where $\mathcal{S}$ is the set of *states*, `inputs` is the set of *input actions*, `outputs` is the set of *output actions*, $\mathcal{I} \subseteq \mathcal{S}$ is the set of *initial states*, and $\mathcal{T}$ is the set of *transitions*. A transition is a tuple $(s, i, o, s') \in \mathcal{T}$ where $s, s' \in \mathcal{S}$ are the *initial* and *final* states, respectively, $i \in \texttt{inputs}$ is the *input* that activates the transition, and $o \in \texttt{outputs}$ is the *output* produced in response. A transition $(s, i, o, s') \in \mathcal{T}$ is also denoted by $s \xrightarrow{i/o} s'$.

We say that $(i_1/o_1, \ldots, i_n/o_n)$ is a *trace* of M if there exists $s_1 \in \mathcal{I}$ and $s_2, \ldots, s_{n+1} \in \mathcal{S}$ such that $s_1 \xrightarrow{i_1/o_1} s_2, s_2 \xrightarrow{i_2/o_2} s_3, \ldots, s_n \xrightarrow{i_n/o_n} s_{n+1}$ are transitions of $\mathcal{T}$. The set of all traces of M is denoted by $\texttt{traces}(M)$. Let us consider $s, s' \in \mathcal{S}$. We say that s' is *reachable* from s, denoted by $\texttt{isReachable}(M, s, s')$, if either there exist u, i, o such that $s \xrightarrow{i/o} u \in \mathcal{T}$ and $\texttt{isReachable}(M, u, s')$ holds, or $s = s'$. The set $\texttt{reachableStates}(M, s)$ contains all $s' \in \mathcal{S}$ such that $\texttt{isReachable}(M, s, s')$.

Let $s \in \mathcal{S}$ and $i \in \texttt{inputs}$. $\texttt{outs}(M, s, i)$ denotes the set of outputs that can be produced in s in response to i, that is, the set $\{o \mid \exists\, s' : s \xrightarrow{i/o} s' \in \mathcal{T}\}$.

We say that $s \in \mathcal{S}$ is *deterministic*, denoted by $\texttt{isDet}(M, s)$, if there do not exist $s \xrightarrow{i/o'} s', s \xrightarrow{i/o''} s'' \in \mathcal{T}$ such that $o' \neq o''$ or $s' \neq s''$. □

In the previous definition, let us note that machines are allowed to be nondeterministic. In order to fix the kind of formalisms our logic will deal with, the following hypothesis will be imposed: Both implementations and specifications can be represented by appropriate `FSMs`. As a consequence, we have that when an input is offered to an IUT it always produces an observable response (that is, *quiescent* states not producing any output are not considered). Next we present

the basic conformance relation that will be considered in our framework. This relation is similar to ioco [15] but in the framework of FSMs. This relation has been used in [11] as a preliminary step to define timed conformance relations. Intuitively, an IUT is conforming if it does not *invent* behaviors for those traces that can be executed by the specification. We will assume that the IUT is *input-enabled*, that is, for all state s and input i there exist o, s' such that $s \xrightarrow{i/o} s'$ belongs to the set of transitions of the IUT. During the rest of the paper, and when no confusion arises, we will assume that the FSM representing a generic specification is given by $spec = (\mathcal{S}_{spec}, \mathtt{inputs}_{spec}, \mathtt{outputs}_{spec}, \mathcal{I}_{spec}, \mathcal{T}_{spec})$.

Definition 2. Let S and I be two FSMs. We say that I *conforms to* S, denoted by $I \,\mathtt{conf}\, S$, if for all $\rho_1 = (i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n/o_n) \in \mathtt{traces}(S)$, with $n \geq 1$, we have $\rho_2 = (i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n/o'_n) \in \mathtt{traces}(I)$ implies $\rho_2 \in \mathtt{traces}(S)$. □

Example 1. A simple example, adapted from [5], will be used along the paper to illustrate our framework. A medical ray beaming system is controlled by using three buttons: A button for charging the machine (a single button press increases the voltage by 10 mV), another one for the beam activation, and the last one for resetting the machine at any time. The system will only charge the machine twice (increasing the voltage up to 20 mV) and it only lets to beam twice. Any further attempt to either increase the charge of the machine or to activate the beaming will be rejected because there is a danger of seriously injuring the patient. The FSM specifying this behavior is depicted in Figure 1 (left) and it is defined as $spec\_ray = (\mathcal{S}_{spec\_ray}, \mathtt{inputs}_{spec\_ray}, \mathtt{outputs}_{spec\_ray}, \mathcal{I}_{spec\_ray}, \mathcal{T}_{spec\_ray})$. We have $\mathcal{S}_{spec\_ray} = \{r, c1, c2, b1, b2\}$, where r denotes the *ready* state, $c1/c2$ denote the states where the beamer has been charged one/two times, and $b1/b2$ denote the states where the first/second beaming is performed. $\mathtt{inputs}_{spec\_ray} = \{br, bc, bb\}$, where $br/bc/bb$ respectively denote that the reset/charging/beaming button has been pressed. $\mathtt{outputs}_{spec\_ray} = \{mr, mc, mb, re\}$, where $mr/mc/mb$ respectively denote that the machine is ready/charging/beaming while re denotes that the command has been rejected. Finally, $\mathcal{I}_{spec\_ray} = \{r\}$, that is, the initial state is *ready*. □

3 Predicates of the Logic

In this section we present the predicates that will be part of $\mathcal{HOTL}$. These predicates allow to represent our knowledge and assumptions about the IUT. In particular, they will allow us to represent the *observations* that we have obtained from the IUT during the preliminary *classical* testing phase. *Observations* denote that, in response to a given sequence of inputs, the IUT produced a given sequence of outputs. Let us remark that if one of the sequences shows a behavior that is forbidden by the specification, then the IUT does not conform to the specification and no further analysis is required, that is, there is no need to apply our logic. As we said before, our notion of observation will be able to include some assumptions about the IUT as well as the observed behavior.

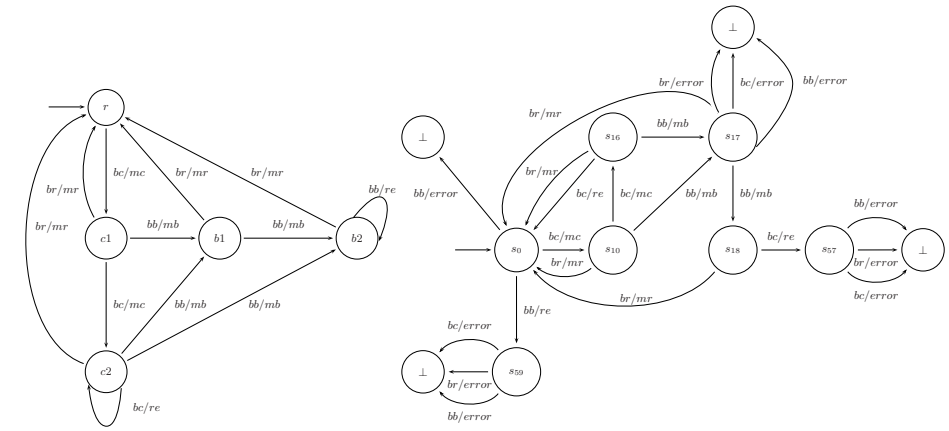

Fig. 1. Finite State Machines $spec\_ray$ (left) and $worst_{spec\_ray}$ (right).

3.1 Manipulating Observations

During the rest of the paper, `Obs` denotes the set of all the observations collected during the preliminary interaction with the IUT, while `Hyp` denotes the set of *hypotheses* the tester has assumed. In this latter set, we will not consider the hypotheses that are implicitly introduced by means of observations.

Observations follow the form $ob = (a_1, i_1/o_1, a_2, \ldots, a_n, i_n/o_n, a_{n+1}) \in \texttt{Obs}$, where ob is a unique identification name. It denotes that when the sequence of inputs $i_1, \ldots, i_n$ was proposed from the initial configuration of the implementation, the sequence $o_1, \ldots, o_n$ was obtained as response. In addition, for all $1 \leq j \leq n+1$, a_j represents a set of *special attributes* concerning the state of the implementation that we reached after performing $i_1/o_1, \ldots, i_{j-1}/o_{j-1}$ in *this* observation. Attributes denote our assumptions about this state. For all $1 \leq j \leq n$ the attributes in the set a_j are of the form $\texttt{imp}(s)$ or `det`, where $\texttt{imp}(s)$ denotes that the state reached after $i_1/o_1, \ldots, i_{j-1}/o_{j-1}$ is associated to a *state identifier* s and `det` denotes that the implementation state reached after $i_1/o_1, \ldots, i_{j-1}/o_{j-1}$ in this observation is deterministic. State identifiers are used to match equal states: If two states are associated with the same state identifier then they represent the *same* state of the implementation.[1] Besides, attributes belonging to a_{n+1} can also be of the form $\texttt{spec}(s)$, with $s \in \mathcal{S}_{spec}$, denoting that the state reached after $i_1/o_1, \ldots, i_n/o_n$ is such that the subgraph that can be reached from it is isomorphic to the subgraph that can be reached from the state s of the specification. We assume that attributes of the form $\texttt{spec}(s)$ can appear only at the end of the observation, meaning that the behavior of the

[1] Let us remark that, since we consider the IUT to be a black-box, a tester cannot always be sure of the state where the IUT is placed. However, she may still *hypothesize* that the reached states after performing two subsequences are in fact the same.

implementation from that point on is known and there is no need to check its correctness.

Example 2. For our case study we will consider that the set of observations $\texttt{Obs}=\{ob_i|1 \leq i \leq 11\}$ was obtained. As illustration, we show the following ones (the full set is given in [14]).

$$\begin{aligned} ob_1 &= (\{det\}, bc/mc, \{imp(q_1)\}, bb/mb, \{imp(q_2)\}, bb/mb, \emptyset) \\ ob_6 &= (\emptyset, bc/mc, \{imp(q_1)\}, bb/mb, \{imp(q_2)\}, bb/mb, \{imp(q_3)\}, br/mr, \emptyset, bc/mc, \\ &\quad \{imp(q_1)\}, bb/mb, \{imp(q_2)\}) \\ ob_{10} &= (\emptyset, bc/mc, \{imp(q_1)\}, bb/mb, \{imp(q_2), det\}, bb/mb, \{imp(q_2)\}, bb/re, \emptyset) \end{aligned}$$

For example, ob_{10} denotes that we initially do not make assumptions about the state the IUT selected as *initial* for this observation (its set of attributes is $\emptyset$). After pressing the charging button, the beaming system is charged and the state reached is identified by a specific state identifier (denoted by q_1). Next, by pressing the beaming button, the beaming is performed and we reach a certain state, in principle different to previous ones, which is assumed to be deterministic and is denoted by the identifier q_2. After pressing again the beaming button, the beaming action is performed and we assume that we reach again the same state as before (note that it is denoted by the same identifier q_2). We press once again the beaming button but this time the action is rejected, and we make no assumptions about the state reached afterwards. □

3.2 Model Predicates

Observations will allow to create *model predicates*. A model predicate denotes our knowledge about the implementation. Models will be constructed according to the observations and hypotheses we consider. In particular, they induce a graph consistent with the observations and hypotheses considered so far. As more information is retrieved, models will be refined and particularized. We denote model predicates by $\texttt{model}\,(m)$, where $m = (\mathcal{S}, \mathcal{T}, \mathcal{I}, \mathcal{A}, \mathcal{E}, \mathcal{D}, \mathcal{O})$. The meaning of the different components of the tuple are the following. $\mathcal{S}$ (*states*) is the set of states that appear in the graph of the model. Despite the fact that this graph attempts to represent (a part of) the behavior of the implementation, any name belonging to $\mathcal{S}$ is fresh and by no means is related to the corresponding state of the implementation. Let us note that after more information is considered, it could turn out that some states belonging to $\mathcal{S}$ coincide. Next, $\mathcal{T}$ (*transitions*) is the set of transitions appearing in the graph of the model. $\mathcal{I}$ (*initial states*) is the set of states that are initial in the model. $\mathcal{A}$ (*accounting*) is the set of *accounting registers*. A register is a tuple $(s, i, outs, n)$ denoting that in the state $s \in \mathcal{S}$ the input i has been offered n times and we have obtained the outputs belonging to the set *outs*. This information allows to handle some hypotheses about nondeterminism. If, due to the hypotheses that we consider, we infer that the number of times we observed an input is enough to believe that the implementation cannot react to that input in a way that has not happened before (that is, either with an output that was not produced before or leading

to a state that was not taken before), then the value n is set to $\top$. In this case, we say that the behavior of the state s for the input i is *closed*. Next, $\mathcal{E}$ (*equality relations*) is the set of equalities relating states in $\mathcal{S}$. Equalities have the form s `is` s'. For example if s_1 `is` $s \in \mathcal{E}$ and s_2 `is` $s \in \mathcal{E}$ then we infer that $s_1 = s_2$ and that one of the names could be eliminated afterwards. $\mathcal{D}$ (*deterministic states*) is the set of states that are deterministic (according to the hypotheses considered so far). Finally, $\mathcal{O}$ (*used observations*) is the set of observations we have used so far for the construction of this model. The aim of recording them is to avoid considering the same observation several times, which could ruin the information codified, for instance, in $\mathcal{A}$.

In $\mathcal{HOTL}$, conclusions about the conformance of a model (that is, of the possible IUTs it represents) with respect to a specification will be established only after the full set of observations `Obs` has been considered. Besides, we will require that no other rule concerning hypotheses in `Hyp` can be applied. In Section 4 we introduce some of the hypotheses a tester might consider in this set. These hypotheses include usual ones such as to assume an upper bound on the number of states of the IUT, the uniqueness of the initial state, the determinism of the IUT, etcetera.

3.3 Other Predicates

We will also consider other predicates related to the correctness of models. The `correct`(m) predicate denotes that m is a correct model, that is, it denotes a behavior that has to be conforming to the specification. The `allModelsCorrect` predicate represents a set of correct models. This predicate is the *goal* of the logic: If it holds then all the IUTs that could produce the observations in `Obs` and meet all the requirements in `Hyp` conform to the specification. The `consistent`(m) predicate means that the model m does not include any *inconsistency*. Note that the requirements imposed by `Obs` and `Hyp` could lead to inconsistent models. For example, let us consider a model where a state s is assumed to be deterministic, s is equal to another state s', and s' produces either o_1 or o_2 when i is offered, with $o_1 \neq o_2$. There is no `FSM` that meets the requirements of this model. Since a user of the logic can create a set of observations and hypotheses leading to that model, inconsistent models may indeed appear. As we will see, the rules of the logic will eliminate inconsistent models by deducing an *empty* set of models from them. In addition, we will be provided with rules that allow to *guarantee* the consistency of a model.

In general, several models can be constructed from a set of observations and hypotheses. Hence, our logic will deal with *sets* of models. If $\mathcal{M}$ is a set of models then the predicate `models`$(\mathcal{M})$ denotes that, according to the observations and hypotheses considered, $\mathcal{M}$ contains all the models that are valid candidates to properly describe the implementation. Besides, `modelsSubset`$(\mathcal{M}')$ denotes that for some set $\mathcal{M}$ we have `models`$(\mathcal{M})$ and $\mathcal{M}' \subseteq \mathcal{M}$.

The formal semantics of predicates, which is defined in terms of the set of `FSMs` that fulfill each predicate, is introduced in [14]. These concepts are considered there to prove the soundness and completeness of the logic.

4 Deduction Rules of the Logic $\mathcal{HOTL}$

The rules will be presented in the form $\frac{premises}{conclusion}$. If B can be deduced from A then we write $A \vdash B$. If we want to be more specific, we write $A \vdash_r B$ to denote that B is deduced from A by applying the rule r. The ultimate goal is to deduce the *conformance* of a set of observations `Obs` and hypotheses `Hyp`, that is, whether *all* the `FSMs` that meet these conditions conform to the specification. Since inconsistent models may appear, conformance will be granted only if there exists at least one *consistent* model that meets these conditions. Some formal definitions and rules could not be included in this version of the paper and can be found in [14]. In these cases, brief informal explanations will be provided.

$\mathcal{HOTL}$ will consider observations and hypotheses in two phases. First, observations, as well as the hypotheses they can implicitly express, will be collected. Once all of them have been considered (i.e., we have a model predicate with $\mathcal{O} = \texttt{Obs}$) a second phase, to add the rest of hypotheses, will start.

First, we present a rule to construct a model from an observation. Given a predicate denoting that an observation was collected, the rule deduces some details about the behavior of the implementation. These details are codified by means of a *model* that shows this behavior. Basically, *new* states and transitions will be created in the model so that it can produce the observation. Even though some model states could actually coincide, we will not consider it yet. Thus, we take fresh states to name all of them. Besides, the hypotheses denoted by the attributes of the observation will affect the information associated to the corresponding model states. In particular, if the tester assumes that the last state of the observation is isomorphic to a state of the specification (i.e., $\texttt{spec}(s)$, for some $s \in \mathcal{S}_{spec}$) then the sets of states, transitions, accounting registers, and deterministic states will be extended with some extra elements taken from the specification and denoted by $\mathcal{S}'$, $\mathcal{T}'$, $\mathcal{A}'$, and $\mathcal{D}'$, respectively. The new states and transitions $\mathcal{S}'$ and $\mathcal{T}'$, respectively, will copy the structure existing among the states that can be reached from s in the specification. The new accounting, $\mathcal{A}'$, will denote that the knowledge concerning the new states is *closed* for all inputs, that is, the only transitions departing from these states are those we copy from the specification and no other transitions will be added in the future. Finally, those model states that are images of deterministic specification states will be included in the set $\mathcal{D}'$ of deterministic states of the model.

$$(obser)\frac{ob = (a_1, i_1/o_1, a_2, \ldots, a_n, i_n/o_n, a_{n+1}) \in \texttt{Obs} \ \wedge \ s_1, \ldots, s_{n+1} \text{ are fresh states}}{\texttt{model}\left(\begin{array}{c} \{s_1, \ldots, s_{n+1}\} \cup \mathcal{S}', \\ \{s_1 \xrightarrow{i_1/o_1} s_2, \ldots, s_n \xrightarrow{i_n/o_n} s_{n+1}\} \cup \mathcal{T}', \{s_1\}, \\ \{(s_j, i_j, \{o_j\}, 1) \mid 1 \leq j \leq n\} \cup \mathcal{A}', \\ \{s_j \ \texttt{is} \ s'_j | 1 \leq j \leq n+1 \ \wedge \ \texttt{imp}(s'_j) \in a_j\}, \\ \{s_j \mid 1 \leq j \leq n+1 \ \wedge \ \texttt{det} \in a_j\} \cup \mathcal{D}', \{ob\} \end{array}\right)}$$

The formal definition of $\mathcal{S}'$, $\mathcal{T}'$, $\mathcal{A}'$, and $\mathcal{D}'$ follows. If there does not exist s' such that $\texttt{spec}(s') \in a_{n+1}$ then $(\mathcal{S}', \mathcal{T}', \mathcal{A}', \mathcal{D}') = (\emptyset, \emptyset, \emptyset, \emptyset)$. Otherwise, that is, if $\texttt{spec}(s) \in a_{n+1}$ for some $s \in \mathcal{S}_{spec}$, let us consider the following set of

states: $U = \{u_j \mid u_j \text{ is a fresh state } \wedge\ 1 \leq j < |\texttt{reachableStates}(spec, s)|\}$ and a bijective function $f : \texttt{reachableStates}(spec, s) \longrightarrow U \cup \{s_{n+1}\}$ such that $f(s) = s_{n+1}$. Then, $(\mathcal{S}', \mathcal{T}', \mathcal{A}', \mathcal{D}')$ is equal to

$$\left(\begin{array}{c} U, \quad \{f(s') \xrightarrow{i/o} f(s'') \mid s' \xrightarrow{i/o} s'' \in \mathcal{T}_{spec} \ \wedge\ \texttt{isReachable}(spec, s, s')\}, \\ \{(u, i, \texttt{outs}(spec, s, i), \top) | u \in U \cup \{s_{n+1}\} \ \wedge\ i \in \texttt{inputs}_{spec}\}, \\ \{f(s') | \texttt{isReachable}(spec, s, s') \ \wedge\ \texttt{isDet}(spec, s')\} \end{array}\right)$$

Example 3. If we apply the *obser* deduction rule to the observation ob_6 given in Example 2 then we obtain a model $m_6 = (\mathcal{S}_6, \mathcal{T}_6, \mathcal{I}_6, \mathcal{A}_6, \mathcal{E}_6, \mathcal{D}_6, \mathcal{O}_6)$, where

$\mathcal{S}_6 = \{s_{26}, s_{27}, s_{28}, s_{29}, s_{30}, s_{31}, s_{32}\}$ and $\mathcal{I}_6 = \{s_{26}\}$

$$\mathcal{T}_6 = \left\{\begin{array}{l} s_{26} \xrightarrow{bc/mc} s_{27}, s_{27} \xrightarrow{bb/mb} s_{28}, s_{28} \xrightarrow{bb/mb} s_{29}, s_{29} \xrightarrow{br/mr} s_{30}, \\ s_{30} \xrightarrow{bc/mc} s_{31}, s_{31} \xrightarrow{bb/mb} s_{32} \end{array}\right\}$$

$$\mathcal{A}_6 = \left\{\begin{array}{l} (s_{26}, bc, \{mc\}, 1), (s_{27}, bb, \{mb\}, 1), (s_{28}, bb, \{mb\}, 1), (s_{29}, br, \{mr\}, 1), \\ (s_{30}, bc, \{mc\}, 1), (s_{21}, bb, \{mb\}, 1) \end{array}\right\}$$

$\mathcal{E}_6 = \{s_{27}$ is q_1, s_{28} is q_2, s_{29} is q_3, s_{31} is q_1, s_{32} is $q_2\}$, $\mathcal{D}_6 = \emptyset$, and $\mathcal{O}_6 = \{ob_6\}$

Similarly, for all $1 \leq i \leq 11$ we can obtain a model m_i by applying the deduction rule *obser* to ob_i. □

We will be able to join different models created from different observations into a single model. The components of the new model will be the union of the components of each model.

$$(\textit{fusion}) \frac{\begin{array}{c} \texttt{model}\ (\mathcal{S}_1, \mathcal{T}_1, \mathcal{I}_1, \mathcal{A}_1, \mathcal{E}_1, \mathcal{D}_1, \mathcal{O}_1) \ \wedge \\ \texttt{model}\ (\mathcal{S}_2, \mathcal{T}_2, \mathcal{I}_2, \mathcal{A}_2, \mathcal{E}_2, \mathcal{D}_2, \mathcal{O}_2) \ \wedge\ \mathcal{O}_1 \cap \mathcal{O}_2 = \emptyset \end{array}}{\texttt{model}\ \left(\mathcal{S}_1 \cup \mathcal{S}_2, \mathcal{T}_1 \cup \mathcal{T}_2, \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{E}_1 \cup \mathcal{E}_2, \mathcal{D}_1 \cup \mathcal{D}_2, \mathcal{O}_1 \cup \mathcal{O}_2\right)}$$

The condition $\mathcal{O}_1 \cap \mathcal{O}_2 = \emptyset$ appearing in the previous rule avoids to include the same observation in a model more than once, which would be inefficient. Besides, since models in the second phase must fulfill $\mathcal{O} = \texttt{Obs}$, we avoid to use the previous rule in the second phase.

By iteratively applying these two first rules, we will finally obtain a model where $\mathcal{O}$ includes all the observations belonging to the set `Obs`.

Example 4. The deduction rule *fusion* allows to join all the models obtained after applying the deduction rule *obser* to the set of observations given in Example 2. After it, we have a new model m_T defined as follows:

$$m_T = \texttt{model}\left(\bigcup_{j=1}^{11} \mathcal{S}_j, \bigcup_{j=1}^{11} \mathcal{T}_j, \bigcup_{j=1}^{11} \mathcal{I}_j, \bigcup_{j=1}^{11} \mathcal{A}_j, \bigcup_{j=1}^{11} \mathcal{E}_j, \bigcup_{j=1}^{11} \mathcal{D}_j, \texttt{Obs}\right)$$

□

At this point, the inclusion of those hypotheses that are covered by observations will begin. During this new phase, in general, we will need several models to represent all the `FSMs` that are compatible with a set of observations and

hypotheses. The next simple rule allows to represent a single model by means of a set containing a single element. Since the forthcoming rules will concern only the second phase, in all cases we will have $\mathcal{O} = \texttt{Obs}$.

$$(set)\frac{\texttt{model}\,(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})}{\texttt{models}\,(\{(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})\})}$$

In order to reflect how a rule that applies to a single model affects the set including this model, we provide the following rule. Let φ denote a logical predicate (in particular, `true`) and $m = \texttt{model}\,(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})$. Then,

$$(propagation)\frac{\texttt{models}\,(\mathcal{M}\cup\{m\})\ \wedge\ \varphi\ \wedge\ ((\texttt{model}\,(m)\ \wedge\ \varphi)\vdash \texttt{modelsSubset}\,(\mathcal{M}'))}{\texttt{models}\,(\mathcal{M}\cup\mathcal{M}')}$$

By using the previous rule, we will be able to use other rules that apply to a *single* model and then propagate its change to the set where the model is included as expected: As the previous rule states, this model changes while other models belonging to the set remain unchanged. Most of the forthcoming rules will apply to single models. After each of them is used, the rule *propagation* will be applied to propagate its effect to the corresponding set of models.

Our logic will allow to discover that a state of the model coincides with another one. In this case, we will eliminate one of the states and will allocate all of its constraints to the other one. This will modify all the components that define the model. This functionality is provided by the `modelElim` function. Specifically, $\texttt{modelElim}(m, s_1, s_2)$ denotes the elimination of the state s_2 and the transference of all its responsibilities to state s_1 in the model m. This function returns a *set* of models. If the transference of responsibilities creates an *inconsistency* in the rest of the model, an empty set of models is returned. Sometimes we will use a *generalized* version of this function to perform the elimination of several states instead of a single state: $\texttt{modelElim}(m, s, \{s_1, \ldots, s_n\})$ represents the substitution of s_1 by s, followed by the substitution of s_2 by s, and so on up to s_n. The formal definitions of both forms of the `modelElim` function are given in [14].

Next we present some rules that use this function. In the first one, we join two states if the set of equalities allows to deduce that both coincide.

$$(equality)\frac{\texttt{model}\,(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})\ \wedge\ \ s_1,s_2\in\mathcal{S}\ \wedge\ \{s_1\ \texttt{is}\ s, s_2\ \texttt{is}\ s\}\subseteq\mathcal{E}}{\texttt{modelsSubset}\,(\texttt{modelElim}((\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs}),s_1,s_2))}$$

Another situation where two states can be fused appears when a deterministic state shows two transitions labelled by the same input. Since the state is deterministic, they must also be labelled by the same output. The determinism of the state implies that both destinations are actually the same state. Hence, these two reached states can be fused. Note that if both outputs are different then the model is inconsistent, because the determinism of the state is not preserved. In this case, an empty set of models is produced.

$$(determ)\frac{\begin{array}{c}\texttt{model}\,(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})\ \wedge\\ s,s_1,s_2\in\mathcal{S}\ \wedge\ s\in\mathcal{D}\ \wedge\ \{s\xrightarrow{i/o_1}s_1, s\xrightarrow{i/o_2}s_2\}\subseteq\mathcal{T}\end{array}}{\texttt{modelsSubset}\,(\mathcal{M}')\ [\text{if } o_1=o_2 \text{ then } \mathcal{M}'=\texttt{modelElim}(m,s_1,s_2) \text{ else } \mathcal{M}'=\emptyset]}$$

Next we present the first rule dealing with an hypothesis that is not implicitly given by an observation. This hypothesis allows to assume that the initial state of the implementation is *unique*.

$$(singleInit)\frac{\texttt{model}\,(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})\ \wedge\ \mathcal{I}=\{s_1,\ldots,s_n\}\ \wedge\ \texttt{singleInitial}\in\texttt{Hyp}}{\texttt{modelsSubset}\left(\texttt{modelElim}((\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs}),s_1,\{s_2,\ldots,s_n\})\right)}$$

If the tester adds the hypothesis that all the states are deterministic, then the complete set of states $\mathcal{S}$ coincides with the set of deterministic states $\mathcal{D}$.

$$(allDet)\frac{\texttt{model}\,(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{D},\texttt{Obs})\ \wedge\ \texttt{allDet}\in\texttt{Hyp}}{\texttt{modelsSubset}\,(\{(\mathcal{S},\mathcal{T},\mathcal{I},\mathcal{A},\mathcal{E},\mathcal{S},\texttt{Obs})\})}$$

The logic $\mathcal{HOTL}$ allows to consider other hypotheses about the IUT. For example, the predicate $\texttt{allTranHappenWith}(n)$ assumes that for all state s and input i such that the IUT behavior has been observed at least n times, *all* the outgoing transitions from s having as input i, have been observed at least once. This means that the IUT state s cannot react to i with an output that has not produced so far or moving to a state it has not moved before. If the hypothesis is assumed then some accounting registers of the model will be set to the value $\top$, denoting that our knowledge about this state and input is *closed*. Depending on the compatibility of the hypothesis with the current model, several models can be produced by this rule. If no model is returned then we infer that the resulting model is inconsistent with the current model requirements. The $\texttt{upperBoundOfStates}(n)$ hypothesis allows to assume that the IUT uses at most n states. The reduction of states, based on the identification of several states with the same state identifiers, will be performed by means of new equalities $s\ \texttt{is}\ s'\in\mathcal{E}$. The $\texttt{longSequencesSamePath}(n)$ hypothesis assumes that if two sequences of n transitions produce the same inputs and outputs, then they actually go through the same states. The set $\mathcal{E}$, containing the assumed equalities between states, will be also used in this case. The formal definition of the rules that allow to consider the $\texttt{allTranHappenWith}(n)$, $\texttt{upperBoundOfStates}(n)$, and $\texttt{longSequencesSamePath}(n)$ hypotheses can be found in [14].

We have seen some rules that may lead to inconsistent models. In some of these cases, an empty set of models is produced, that is, the inconsistent model is eliminated. Before granting conformance, we need to be sure that at least one model belonging to the set is consistent. Next we provide a rule that labels a model as consistent. Let us note that the inconsistences created by a rule can be detected by the forthcoming applications of rules. For instance, the *determ* rule can detect that a previous rule matched a deterministic state with another state in such a way that both react to the input i with a different output. Actually, all inconsistencies can be detected by applying suitable rules. Thus, a model is free of inconsistencies if for any other rule either it is not applicable to the model or the application does not modify the model (i.e., it deduces the same model). Next we introduce this concept. In the following definition, $\mathcal{R}$ denotes the set of all rules in $\mathcal{HOTL}$ that follow the form required to apply the *propagation* rule.

In particular, it consists of all previous rules from *equality* up to the forthcoming *correct* rule.

Definition 3. We denote the set of all rules in $\mathcal{HOTL}$ that follow the form $(\texttt{model}\,(m)\ \wedge\ \varphi) \vdash \texttt{modelsSubset}\,(\mathcal{M}')$ by $\mathcal{R}$.

Let $r = (\texttt{model}\,(m')\ \wedge\ \varphi) \vdash \texttt{modelsSubset}\,(\mathcal{M}') \in \mathcal{R}$ an m be a model. The *unable predicate* for m and r, denoted by $\texttt{unable}(m, r)$, is defined by the expression $\texttt{unable}(m, r) = \neg\varphi\ \vee\ ((\texttt{model}\,(m)\ \wedge\ \varphi) \vdash_r \texttt{modelsSubset}\,(\{m\}))$. We extend this predicate to deal with sets of rules as follows: $\texttt{unable}(m, Q) = \bigwedge\{\texttt{unable}(m, r) | r \in Q\}$. □

The next rule detects that a model is consistent. It requires that no other rule that manages hypotheses can modify the model. These rules consist of all the rules in $\mathcal{R}$ we have seen so far.

$$(consistent)\frac{\begin{array}{c} m = (\mathcal{S}, \mathcal{T}, \mathcal{I}, \mathcal{A}, \mathcal{E}, \mathcal{D}, \texttt{Obs})\ \wedge\ \texttt{model}\,(m)\ \wedge \\ \texttt{unable}(m, \mathcal{R} \backslash \{consistent, correct\}) \end{array}}{\texttt{modelsSubset}\,(\{\texttt{consistent}(\mathcal{S}, \mathcal{T}, \mathcal{I}, \mathcal{A}, \mathcal{E}, \mathcal{D}, \texttt{Obs})\})}$$

Since a model is a (probably incomplete) representation of the IUT, in order to check whether a model conforms to the specification, two aspects must be taken into account. First, only the conformance of consistent models will be considered. Second, given a consistent model, we will check its conformance with respect to the specification by considering the *worst* instance of the model, that is, if this instance conforms to the specification then any other instance extracted from the model does so. This worst instance is constructed as follows: For each state s and input i such that the behavior of s for i is not closed *and* either s is not deterministic or no transition with input i exists in the model, a new *malicious* transition is created. The new transition is labelled with a special output *error*, that does not belong to $\texttt{outputs}_{spec}$. This transition leads to a new state $\perp$ having no outgoing transitions. Since the specification cannot produce the output *error*, this worst instance will conform to the specification only if the unspecified parts of the model are not relevant for the correctness of the IUT it represents.

Definition 4. Let $m = (\mathcal{S}, \mathcal{T}, \mathcal{I}, \mathcal{A}, \mathcal{E}, \mathcal{D}, \texttt{Obs})$ be a model. We define the *worst instance* of the model m with respect to the considered specification *spec*, denoted by $\texttt{worstCase}(m)$, as the FSM

$$\left(\begin{array}{l} \mathcal{S} \cup \{\perp\}, \texttt{inputs}_{spec}, \texttt{outputs}_{spec} \cup \{error\}, \\ \mathcal{T} \cup \left\{ s \xrightarrow{i/error} \perp \;\middle|\; \begin{array}{l} s \in \mathcal{S}\ \wedge\ i \in \texttt{inputs}_{spec}\ \wedge \\ \nexists\, outs : (s, i, outs, \top) \in \mathcal{A}\ \wedge \\ (s \notin \mathcal{D}\ \vee\ \nexists\, s', o : s \xrightarrow{i/o} s') \end{array} \right\}, \mathcal{I} \end{array}\right)$$

□

Thus, the rule for indicating the correctness of a model is

$$(correct)\frac{m = (\mathcal{S}, \mathcal{T}, \mathcal{I}, \mathcal{A}, \mathcal{E}, \mathcal{D}, \texttt{Obs})\ \wedge\ \texttt{consistent}(m)\ \wedge\ \texttt{worstCase}(m)\ \texttt{conf}\ spec}{\texttt{modelsSubset}\,(\{\texttt{correct}(m)\})}$$

Now we can consider the conformance of a set of models. A set conforms to the specification if all the elements do so and the set contains at least one element. Note that an empty set of models denotes that all the models were inconsistent. Hence, granting the conformance of an empty set would imply accepting models that do not represent any implementation. In fact, although *false implies anything*, accepting inconsistent models is useless for a tester.

$$(allCorrect)\frac{\texttt{models}\,(\mathcal{M}) \;\wedge\; \mathcal{M} \neq \emptyset \;\wedge\; \mathcal{M} = \{\texttt{correct}(m_1), \ldots, \texttt{correct}(m_n)\}}{\texttt{allModelsCorrect}}$$

Example 5. We consider the model m_R obtained after applying the *determ*, *equality*, *long* (see [14]), and *singleInit* deduction rules. The *long* rule is applied to introduce the hypothesis `longSequencesSamePath`(1). The *singleInit* and *long* rules are applied once, while *determ* and *equality* are applied as long as we can. Let us recall that, after each of these rules is used, the *propagation* rule must be applied as well. When the *determ* and *equality* rules cannot be applied anymore, our model cannot be further manipulated to produce new inconsistencies. Then, we can use the *consistent* and *propagation* rules to deduce $\texttt{models}\,(\{\texttt{consistent}(m_R)\})$.

We build an `FSM` by applying the function `worstCase` to m_R and we verify its conformance with respect to the specification. The obtained `FSM`, denoted by $worst_{spec\_ray}$, is graphically depicted in Figure 1 (right). For the sake of clarity, we have included four states $\perp$, even though they correspond to only one state.

We have $worst_{spec\_ray}$ `conf` $spec\_ray$ and, by successively applying the *correct* and *propagation* rules, we obtain $\texttt{models}\,(\{\texttt{correct}(m_R)\})$ and deduce, by means of the *allCorrect* deduction rule, `allModelsCorrect`. A more detailed description of the application of rules to this example can be found in [14]. □

Now that we have presented the set of deduction rules, we introduce a correctness criterion. In the next definition, in order to uniquely denote observations, fresh names are assigned to them. Besides, let us note that all hypothesis predicates follow the form $h \in \texttt{Hyp}$ for some h belonging to `Hyp`.

Definition 5. Let *spec* be an `FSM`, `Obs` be a set of observations, and `Hyp` be a set of hypotheses. Let $A = \{ob = o \mid ob \text{ is a fresh name } \wedge\ o \in \texttt{Obs}\}$ and $B = \{h_1 \in \texttt{Hyp}, \ldots, h_n \in \texttt{Hyp}\}$, where $\texttt{Hyp} = \{h_1, \ldots, h_n\}$. If the deduction rules allow to infer `allModelsCorrect` from the set of predicates $C = A \cup B$, then we say that C *logically conforms to spec* and we denote it by $C\ \texttt{logicConf}\ spec$. □

In order to prove the validity of our method, we have to relate the deductions that we make by using our logic with the notion of conformance introduced in Definition 2. The *semantics* of a logic predicate is described in terms of the set of `FSMs` that fulfill the requirements given by the predicate; given a predicate p, we denote this set by $\nu(p)$. As illustration, the semantics of some predicates is formally defined in [14] by means of the function ν. Let us consider that P is the conjunction of all the considered observation and hypothesis predicates.

Then, the set $\nu(P)$ denotes all the FSMs that can produce these observations and fulfill these hypotheses, that is, it denotes all the FSMs that, according to our knowledge, can *define* the IUT. So, if our logic deduces that all of these FSMs conform to the specification (i.e., allModelsCorrect is obtained), then the IUT actually conforms to the specification.

Theorem 1. Let *spec* be an FSM and C be a set of predicates including at least one observation predicate. Then, C logicConf *spec* iff for all FSM $f \in \nu(\bigwedge_{p \in C})$ we have f conf *spec* and $\nu(\bigwedge_{p \in C}) \neq \emptyset$. □

Corollary 1. Let IUT and *spec* be FSMs and C be a set of predicates including at least one observation predicate. If $IUT \in \nu(\bigwedge_{p \in C})$ then C logicConf *spec* implies IUT conf *spec*. If there exists $f \in \nu(\bigwedge_{p \in C})$ such that f conf *spec* does not hold then C logicConf *spec* does not hold.

5 Conclusions and Future Work

In this paper we have presented a logic to infer whether a collection of observations obtained by testing an IUT together with a set of hypotheses allow to deduce that the IUT conforms to the specification. A repertory of heterogeneous hypotheses providing a tester with expressivity to denote a wide range of testing scenarios has been presented. By considering those observations and hypotheses that better fit into her necessities, the tester can obtain diagnosis results about the conformance of an IUT in a flexible range of situations. Besides, our logic allows her to iteratively add observations (i.e., the results of the application of tests) and/or hypotheses until the complete set of predicates guarantees the conformance. In this sense, our logic can be used to dynamically *guide* the steps of a testing methodology.

As future work, we will study some ways to improve our logic. We plan to include an *incorrectness* rule, that is, a rule that detects whether a model is *necessarily incorrect*. If an incorrect model is detected then the calculus can be early terminated, which improves the efficiency. Moreover, the rule could be used to detect which observations/hypotheses made the model incorrect. Besides, we want to develop a more complex application example in the context of Internet protocols. We would also like to introduce a *feasibility* score for each of the logic rules. For example, for a given framework, we can consider that assuming that all the states are deterministic is harder than assuming that the implementation has less that 50 states. In this case, a lower feasibility score will be assigned to the first hypothesis. By accounting the feasibility of all the hypotheses that we have to add before ensuring conformance, we will obtain a measure of the suitability of the considered observations and, indirectly, of the tests that we used to obtain them. Hence, our logic can help a tester to choose her tests so that more *trustable* diagnosis results are obtained. We also consider to extend the repertory of hypotheses. Finally, we want to extend the logic so that it can

deal with *extended finite state machines*. In this case, different formalisms to work with models and different sets of hypotheses will be considered.

Acknowledgements We would like to thank the anonymous referees for their valuable comments. Though they proposed very interesting ideas to improve our paper (some are commented above), we could not apply all of them due to the lack of space. Certainly, these ideas will be considered in the future.

References

1. E. Bayse, A. Cavalli, M. Núñez, and F. Zaïdi. A passive testing approach based on invariants: Application to the WAP. *Computer Networks*, 48(2):247–266, 2005.
2. B.S. Bosik and M.U. Uyar. Finite state machine based formal methods in protocol conformance testing. *Computer Networks & ISDN Systems*, 22:7–33, 1991.
3. A. Cavalli, C. Gervy, and S. Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Journal of Information and Software Technology*, 45:837–852, 2003.
4. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
5. G. Eleftherakis and P. Kefalas. Towards model checking of finite state machines extended with memory through refinement. In *Advances in Signal Processing and Computer Technologies*, pages 321–326. World Scientific and Engineering Society Press, 2001.
6. R. Hierons. Comparing test sets and criteria in the presence of test hypotheses and fault domains. *ACM Transactions on Software Engineering and Methodology*, 11(4):427–448, 2002.
7. D. Lee, D. Chen, R. Hao, R. Miller, J. Wu, and X. Yin. A formal approach for passive testing of protocol data portions. In *10th IEEE Int. Conf. on Network Protocols, ICNP'02*, pages 122–131. IEEE Computer Society Press, 2002.
8. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
9. L.P. Lima and A. Cavalli. A pragmatic approach to generating tests sequences for embedded systems. In *10th Workshop on Testing of Communicating Systems*, pages 288–307. Chapman & Hall, 1997.
10. S.C. Ntafos. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering*, 14:868–874, 1988.
11. M. Núñez and I. Rodríguez. Encoding `PAMR` into (timed) `EFSMs`. In *FORTE 2002, LNCS 2529*, pages 1–16. Springer, 2002.
12. A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School, MOVEP 2000, LNCS 2067*, pages 196–205. Springer, 2001.
13. A. Petrenko, N. Yevtushenko, and G. von Bochmann. Testing deterministic implementations from their nondeterministic specifications. In *9th Workshop on Testing of Communicating Systems*, pages 125–140. Chapman & Hall, 1996.
14. I. Rodríguez, M.G. Merayo, and M. Núñez. A logic for assessing sets of heterogeneous testing hypotheses: Extended version. Available at: `http://dalila.sip.ucm.es/~manolo/papers/logic-extended.pdf`, 2006.
15. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.