Conformance Testing of Boolean Programs with Multiple Faults

Pavithra Prabhakar[1,2] * and Mahesh Viswanathan[3]

[1] California Institute of Technology
[2] IMDEA Software Institute
[3] University of Illinois at Urbana-Champaign

Abstract. Conformance testing is the problem of constructing a complete test suite of inputs based on a specification S such that any implementation I (of size less than a given bound) that is not equivalent to S gives a different output on the test suite than S. Typically I and S are assumed to be some type of finite automata. In this paper we consider the problem of constructing test suites for boolean programs (or more precisely modular visibly pushdown automata) that are guaranteed to catch all erroneous implementations that have at least R faults, and pass all correct implementations; if the incorrect implementation has fewer than R faults then the test suite may or may not detect it. We present a randomized algorithm for the construction of such test suites, and prove the near optimality of our test suites by proving lower bounds on the size of test suites.

1 Introduction

Conformance testing is the problem of designing test suites based on a given formal specification S which is typically an automaton (either finite or infinite). In the framework of conformance testing in general, the implementation I being tested, is assumed to have an unknown internal structure, but can be tested by applying a sequence of inputs and observing the outputs it produces. Given an integer bound N, the goal is to construct a test suite T such that if some I of size less than N does not "conform" to S, then there is some input sequence in T on which the outputs of I and S differ. Typically, the notion of conformance is taken to be language equivalence, though weaker notions such as ioco have been explored [13]. Such conformance tests have not only been used to test circuits and protocols [6, 10] but have also been used to model check black-box systems [12, 5].

Since Moore's seminal work [11] on this problem, many algorithms for solving conformance testing have been proposed; major results are summarized in [6, 4, 9, 10] [4]. All of these algorithms construct test suites when the specification and

* This work was done while the first author was a student at the University of Illinois at Urbana-Champaign.

[4] These references are to algorithms that construct complete test suites, which is the focus of this paper. There has also been a lot of work on constructing incomplete test suites that catch all bugs in the limit.

implementation are assumed to be finite state automata. Broadly, it is understood that the running time of the algorithm and the size of the test suite are polynomial in the size of the specification, when the implementation is assumed to have at most as many states as the specification. When the implementation can have Δ extra states, the running time and the size of the constructed test suite have an exponential dependence on Δ. These bounds are known to be optimal [14].

While finite state models are convenient abstractions in many situations, in order to faithfully model software, it becomes imperative to consider models that explicitly capture recursion. Therefore, Boolean programs, or more precisely *modular VPAs* [7], have been considered, and the results on conformance testing finite state machines have been extended to such recursive models [7]. Modular VPAs are an automata model, inspired by Visibly Pushdown Automata (VPA) [2] and Recursive State Machines (RSMs) [1], that capture sequential recursive programs all of whose data variables are Boolean. Thus, they are pushdown automata whose control states have been partitioned into modules that correspond to functions in a program. The trace of these machines explicitly encodes recursive function calls by the name of the module being called and the associated parameter, as well as returns from such calls. In addition, like in typical programming languages, function calls result in the calling state being pushed onto the call stack; the parameter of the call is not pushed onto the stack but is rather stored in the local state of the called module. These restrictions ensure that modular VPAs have unique minimized (in terms of the number of control states) machines that can be constructed in polynomial time. The minimization procedure is based on a congruence-based characterization of modular VPAs, which can then be exploited to solve the conformance testing problem. Assuming that the specification S is a minimized modular VPA and the number of control states of the implementation I is not more than that of S, there is polynomial time algorithm that constructs a complete test suite. The input sequences in the test suite are presented symbolically using an equation system; when expanded to get an explicit sequence of input symbols, the size can be exponential in the number of control states of S. This exponential dependency cannot be avoided because the shortest path reaching a particular control state in a pushdown system can be exponentially long. When the implementation I has Δ additional control states, the running time of the algorithm, the symbolic representation of the test suite, and the explicit representation of the test suite, are exponentially dependent on Δ.

In this paper we investigate if the exponential dependence on the extra states of the implementation can be avoided if we relax the completeness requirements of the test suite. More precisely, we consider the problem of designing an (R, Δ) conformance test. An (R, Δ) test for a specification S with n control states, is a test suite T such that any implementation I that has at most $n + \Delta$ control states and at least R faults, gives a different output from S on some input in T; here we say that I has at least R faults, if at least R changes to the transition relation of I must be made in order to get a correct implementation. The notion

of (R, Δ) conformance tests was first introduced in [8], where such test suites were constructed for specifications and implementations that are finite state machines. In this paper, we continue this line of work, and extend it to the case of recursive software.

In order to explain the challenges and contributions of our work, we recall the main ideas used in conformance testing algorithms. In a minimized specification machine S, any pair of control states p, q can be *distinguished* by a test; in the case of finite state systems it is simply an input sequence that is applied from p and q, and in the case of pushdown systems, it is a pair of input strings — one that sets up a common stack for p and q, and the other on which p and q give different outputs. Moreover, for every control state q, there is an input string, called the *access string* for q, that takes the specification to q. When the implementation does not have extra states, the idea is to check that the implementation state reached on x_q (access string for q) "behaves like" q; in other words, this implementation state gives the same output as q on all the distinguishing tests, and transitions out of the implementation state go to states that behave like the target of transitions out of q. When there are Δ extra states, the test suite must have input sequences that visit the states not reached by the x_q inputs (called "unknown" states of I), and check the transitions out of those.

If I and S are finite state machines, then these unknown states can be reached within Δ transition steps from a "known state". Thus, the idea is to have tests that explore *every* input sequence of length at most Δ from *every* known state, and check that the states reached in I behave the same way as the states reached in S after the same input sequence. Hence, both the size of the test and the time to generate it, depend exponentially on Δ. Moreover, if any of these "walks" of length at most Δ from known states is omitted from the test, then the test suite cannot guarantee to catch every incorrect implementation. When the completeness requirements are relaxed (namely, to provably catch implementations with at least R faults), two factors come into play [8]. First, one can show that a "faulty" state can be reached in a fewer number of steps from a known state, when I has at least R faults; here, by faulty state we mean one for which the error can be observed when the distinguishing tests are applied. Second, many of these short walks from known states lead to faulty states. Thus, if we were to choose (randomly) a few of these short walks from every known state, then the test suite is likely to catch every implementation with at least R faults. These observations were used in [8] to give a randomized algorithm that outputs a small test suite that, with high probability, is likely to be a (R, Δ) conformance test.

When I and S are recursive programs, the situation changes. Since the shortest path reaching a particular control state in a pushdown automaton can be exponentially long, this means that unknown states may be reached only if we take $2^{n+\Delta}$ steps from a known state. Thus, a naïve application of the observations from the finite state case suggests that the dependence of the size of the test suite on Δ would be doubly exponential. However, it was observed in [7] that one doesn't need to consider all input sequences of length $2^{n+\Delta}$ from known

states, but rather only certain "special" ones that are described succinctly using equation systems of linear size. This key observation was used to get a test generation algorithm and test suite, whose asymptotic complexity is similar to that for finite state systems.

For (R, Δ) conformance tests and modular VPAs, the ideas from the finite state case do not extend easily. In [8], the proof of the existence of many short walks from known states to faulty states, relied on the existence of large cuts separating known states from faulty states in the implementation. Such large cuts do not seem to exist for modular VPAs. However, using new proof techniques, we show that when I has R faults, many paths described by equations systems lead to faulty states. Thus, if the test generation algorithm randomly picks some of these walks from known states then, with high probability, the resulting test suite will be a (R, Δ) conformance test. Note that, unlike the finite state case, we cannot show that "short walks" are sufficient. Finally, we present lower bounds on the size of (R, Δ) conformance tests for modular VPAs. These lower bounds demonstrate that the test suite constructed by our randomized algorithm is close to optimal.

We conclude this introduction by discussing the practical relevance of our algorithm, and (R, Δ) tests in general. Our algorithm is a randomized algorithm that is highly likely to output a (R, Δ) test. Here the probability of error is over random decisions made by the algorithm, not on a distribution over machines S and I. Thus the error can be reduced to as small a number as desired by increasing the size of the test suite. (R, Δ) tests, though guaranteed to catch implementations with at least R faults, nonetheless, can detect errors in implementations with fewer faults. Thus, (R, Δ) tests can be seen as incomplete tests along a dimension orthogonal to traditional metrics like coverage. Therefore, their importance is derived not so much in the precise way we count faults in an implementation, but rather from the fact that their incompleteness can be mathematically characterized. Hence, (R, Δ) tests should be seen as a hierarchy of test suites of increasing precision, to be chosen from, based on practical time constraints imposed on the testing process by product release times.

2 Modular Visibly Pushdown Automata ($MVPA$)

Boolean programs are essentially programs in any imperative language in which all the variables have a boolean type. In particular, they do not have dynamic memory, and parameters are passed to functions by call-by-value. Formally, they define a Modular Visibly Pushdown Automata, which we present next.

Modular Visibly Pushdown Automata. Let M be a finite set of *modules* and $m_0 \in M$ be the initial module. For each $m \in M$, let P_m be a nonempty finite set of *parameters* and let $P_{m_0} = \{p_0\}$. A *call* c is a pair (m, p) where $m \in M \backslash \{m_0\}$ and $p \in P_m$, and denotes the action of calling a module m with parameter p (we won't allow initial module to be called except at the beginning, and hence (m_0, p_0) will not be a call). Let Σ^{call} denote the set of all calls. Let Σ^{int} be

a finite set of internal actions, and let $\Sigma^{ret} = \{r\}$ be the alphabet of returns, containing the unique symbol r. We will assume that the sets Σ^{call}, Σ^{int} and Σ^{ret} are mutually disjoint. Let $\hat{\Sigma} = (\Sigma^{call}, \Sigma^{int}, \Sigma^{ret})$ and let $\Sigma = \Sigma^{call} \cup \Sigma^{int} \cup \Sigma^{ret}$. We call $\langle M, \{P_m\}_{m \in M}, m_0, \hat{\Sigma} \rangle$ a *signature*.

A *modular visibly pushdown automaton (MVPA)* $\mathcal{A}$ over the signature $\langle M, \{P_m\}_{m \in M}, m_0, \hat{\Sigma} \rangle$ is a tuple $(\{Q_m, \{q_m^p\}_{p \in P_m}, \delta_m\}_{m \in M}, F)$ where for each $m \in M$:

- Q_m is a finite set of states. We assume that for $m \neq m'$, $Q_m \cap Q'_m = \emptyset$. Let $Q = \cup_{m \in M} Q_m$ denote the set of all states.
- For each parameter $p \in P_m$, $q_m^p \in Q_m$ is a state associated with p; we will call this the entry associated with the call (m, p). (Note that we do not insist that q_m^p is different from $q_m^{p'}$, when $p \neq p'$.)
- $\delta_m : Q_m \times (\Sigma^{call} \cup \Sigma^{int} \cup Q) \to Q$ such that:
  - Call transitions:
    for every $q \in Q_m$, $(n, p) \in \Sigma^{call}$, $\delta_m(q, (n, p)) = q_n^p$;
  - Internal transitions:
    for every $q \in Q_m$, $a \in \Sigma^{int}$, $\delta_m(q, a) \in Q_m$;
  - Return transitions:
    for every $q \in Q_m$ and $q' \in Q_{m'}$, $\delta_m(q, q') \in Q_{m'}$;

  The transition function $\delta : Q \times (\Sigma^{call} \cup \Sigma^{int} \cup Q)$ is such that δ restricted to $Q_m \times (\Sigma^{call} \cup \Sigma^{int} \cup Q)$ is δ_m.
- $F \subseteq Q_{m_0}$ is the set of final states.

Notation We write $q \xrightarrow{a}_{\mathcal{A}} q'$ to denote $\delta(q, a) = q'$ for $a \in (\Sigma^{call} \cup \Sigma^{int} \cup Q)$. We drop the subscript $\mathcal{A}$ when it is clear from the context. Unless stated otherwise, we will always assume the signature to be $Sig = \langle M, \{P_m\}_{m \in M}, m_0, \hat{\Sigma} \rangle$.

We consider *MVPA* with deterministic transitions, since the non-deterministic version is equivalent in expressiveness to the deterministic version [7].

A *MVPA* reads words that are well-matched. A word u is a well matched word if the number of occurrences of the call symbols in it is equal to the number of occurrences of return symbols in it, and the number of occurrences of the call symbols in any prefix of u is greater than or equal to the number of occurrences of return symbols in the prefix. The set of all well-matched words over $\hat{\Sigma}$ is denoted by $WM(\hat{\Sigma})$. From now on, we will use u, u', u_i to denote words and w, w', w_i to denote well-matched words.

A *MVPA* operates by reading a well-matched word, and modifying its state and stack accordingly. It starts in the initial state, $q_{m_0}^{p_0}$, with an empty stack. It reads a symbol of the word and takes one of the transitions out of the current state which matches the symbol. When the automaton reads the symbol r, it takes a return transition. It changes its state to the target state of the transition. When an internal transition is taken, the stack remains unchanged. If it takes a call transition, then it pushes the current state onto the stack. A return transition can be taken only if the transition label and the top of the stack match, in which case the top element of the stack is popped.

Formally, the semantics of *MVPA* is defined in terms of a graph over configurations. A *configuration* is a pair $(q, \sigma) \in Q \times Q^*$, where q denotes the current

state of the *MVPA* and σ denotes its stack contents. We assume that the last symbol of σ is the top of the stack. Let *Conf* denote the set of all configurations along with a special configuration c_0. The semantics of a *MVPA* $\mathcal{A}$ is given by a graph (V, E) where the set of vertices V is given by the set of configurations *Conf* and the set of edges $E \subseteq V \times V$ is the smallest set satisfying the following:

- (Initial) The edge $c_0 \xrightarrow{(m_0,p_0)} q_{m_0}^{p_0}$ is in E.
- (Internal) If $(q, \sigma) \in V$, $a \in \Sigma^{int}$ and $\delta(q, a) = q'$, then the edge $(q, \sigma) \xrightarrow{a} (q', \sigma)$ is in E.
- (Call) If $(q, \sigma) \in V$ and $(m, p) \in \Sigma^{call}$, then $(q, \sigma) \xrightarrow{(m,p)} (q_m^p, \sigma q)$ is in E.
- (Return) If $(q, \sigma q') \in V$ and $\delta(q, q') = q''$, then $(q, \sigma q') \xrightarrow{r} (q'', \sigma)$ is in E.

A *run* of $\mathcal{A}$ on a word $u = a_1 \cdots a_n$ is a path in the configuration graph on u, that is, a path $\pi = c_0 c_1 \cdots c_n$ such that $c_i \xrightarrow{a_{i+1}} c_{i+1}$ for all $0 \leq i < n$. Note that such a path is unique. We say that $\mathcal{A}$ *reaches* the state q on u, if there exists a run $\pi = c_0 \cdots c_n$ of $\mathcal{A}$ on u such that $c_n = (q, \sigma)$ for some stack configuration σ. π is an *accepting* run of $\mathcal{A}$ on u if the last configuration c_n of π is (q, σ) for some final state $q \in F$. A word u is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on u. The *language* of $\mathcal{A}$, $L(\mathcal{A})$, is defined as the set of words $u \in \Sigma^*$ accepted by $\mathcal{A}$.

Remark 1. We note that a *MVPA* accepts only well-matched words since the start state $q_{m_0}^{p_0}$ and the final states are all in Q_{m_0}, and every return transition returns to the module of the corresponding call transition.

Example 1. Figure 1 shows an *MVPA* with two modules m_0 and m, with $P_{m_0} = \{p_0\}$ and $P_m = \{p\}$. Here $\Sigma^{int} = \{a, b\}$, $q_{m_0}^{p_0} = q_0$, $q_m^p = q_1$ and $F = q_0$. There are internal transitions within the module and return transitions from module m to module m_0 on states of m_0. There are call transitions from every state to q_1 on (m, p). The language of the *MVPA* in Figure 1 is the set of all well matched words without nested calls such that any non-empty sequence between a call and the following return consists of an alternating sequence of as and bs starting with an a.

Later we will need the notion of a partial homomorphism and homomorphism, which we define below:

Definition 1. *Given MVPA* $\mathcal{A} = (\{Q_m, \{q_m^p\}_{p \in P_m}, \delta_m\}_{m \in M}, F)$ *and* $\mathcal{A}' = (\{{Q'}_m, \{{q'}_m^p\}_{p \in P_m}, {\delta'}_m\}_{m \in M}, F')$ *over the signature Sig with* $Q = \cup_{m \in M} Q_m$ *and* $Q' = \cup_{m \in M} {Q'}_m$*, a function* $h : Q'' \to Q$*, where* Q'' *is a subset of* Q' *containing* ${q'}_{m_0}^{p_0}$*, is called a* partial homomorphism *from* $\mathcal{A}'$ *to* $\mathcal{A}$ *if:*

B1 $h({q'}_{m_0}^{p_0}) = q_{m_0}^{p_0}$.
B2 For every $q' \in Q''$*,* $q' \in F'$ *iff* $h(q') \in F$.
B3 For every $q' \in Q''$ *and* $a \in \Sigma^{call} \cup \Sigma^{int} \cup Q''$*, if* $\delta'(q', a) \in Q''$*, then* $h(\delta'(q', a)) = \delta(h(q'), a')$*, where* $a' = a$ *if* $a \in \Sigma^{call} \cup \Sigma^{int}$ *and* $a' = h(a)$ *otherwise.*

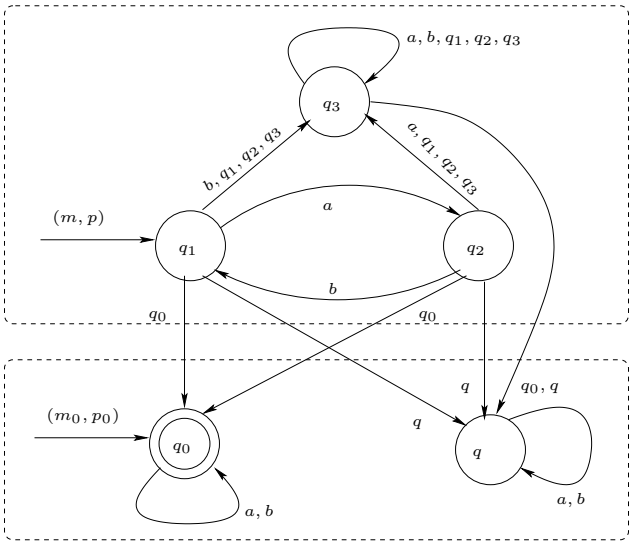

Fig. 1. Example *MVPA*: There are call transitions from every state to q_1 on (m, p) which is omitted from the figure.

In addition, if the following condition is satisfied, then we call h a *homomorphism.*

B4 For every $q' \in Q''$ and $a \in \Sigma^{call} \cup \Sigma^{int} \cup Q''$, $\delta'(q', a) \in Q''$.

Proposition 1 *If there is a homomorphism from $\mathcal{A}'$ to $\mathcal{A}$, then $\mathcal{A}'$ and $\mathcal{A}$ are* equivalent*, that is, $L(\mathcal{A}') = L(\mathcal{A})$.*

Next we state a result from [7] on the minimization of *MVPA*. Since the size of the *MVPA*, which is the total number of bits required to describe it, is polynomial in the number of states of the automaton, we will consider the size of the automaton to be the number of states.

Theorem 1 ([7]). *Given a MVPA $\mathcal{A}$ over Sig, there exists a unique minimal MVPA $\mathcal{A}_{min}$ over Sig such that $L(\mathcal{A}_{min}) = L(\mathcal{A})$.*

The construction of the automaton $\mathcal{A}_{min}$ is based on the following congruence relation. For each $m \in M$, the equivalence relation $\sim_m$ on $P_m \times WM(\hat{\Sigma})$ which depends on $L = L(\mathcal{A})$ (and not on $\mathcal{A}$) is defined as follows: $(p_1, w_1) \sim_m (p_2, w_2)$ iff $\forall u, v \in \Sigma^*$, $u(m, p_1)w_1v \in L \Leftrightarrow u(m, p_2)w_2v \in L$. The states of $\mathcal{A}_{min}$ in the module m are the equivalence classes of $\sim_m$, and a state $[(p, w)]$ can be reached by the string $(m_0, p_0)(m, p)w$. From the definition of $\sim_m$, it is clear that given two distinct states $[(p_1, w_1)]$ and $[(p_2, w_2)]$ of module m, there exist $u, v \in \Sigma^*$ such that exactly one of $u(m, p_1)w_1v$ and $u(m, p_2)w_2v$ is in L. We call (u, v) a distinguishing pair and $(m, p_1)w_1$ an access string. We will need these notions later, hence we will define these next.

Informally an access string is a word which can be used to reach a certain state from an entry state of its module. An *access string* for a state q of module m is a string of the form $(m,p)w$, where $p \in P_m$ and $w \in WM(\hat{\Sigma})$, such that $(m_0,p_0)(m,p)w$ reaches q. We call a state *accessible* if there is an access string x such that $(m_0,p_0)x$ reaches it. A *complete set of access strings* is a set containing an access string for every accessible state of the automaton. Given a *MVPA* $\mathcal{A}$ and an access string x, we denote the state reached in $\mathcal{A}$ on $(m_0,p_0)x$ by $state_{\mathcal{A}}(x)$.

For distinct states q_1, q_2 in module m of $\mathcal{A}$, a pair of strings (u,v) is a *distinguishing test* for $\{q_1,q_2\}$ if for all access strings $(m,p_1)w_1$ and $(m,p_2)w_2$ of q_1 and q_2 respectively, exactly one of $u(m,p_1)w_1v$ and $u(m,p_2)w_2v$ is in $L(\mathcal{A})$. We also say that (u,v) distinguishes q_1 and q_2. D is a *complete set of distinguishing* tests if for every module m and distinct states q_1, q_2 in module m of $\mathcal{A}$, there is a distinguishing test $(u,v) \in D$ for $\{q_1,q_2\}$. Observe that a complete set of distinguishing tests always exists for a minimal *MVPA*.

For the *MVPA* in Figure 1, an access string for q_2 is $(m,p)a$, and an access string for q is $a(m,p)br$. A distinguishing test for the states q_1, q_2 is $u = (m_0,p_0)$ and $v = ar$, since for any access string x for q_1, uxv belongs to the language of the *MVPA* and for any access string y for q_2, uyv does not belong to the language.

Let $\mathcal{A}$ be a minimal *MVPA* with n states and $\{x_1, \cdots, x_n\}$ be a complete set of access strings for $\mathcal{A}$ (every state of a minimal *MVPA* is accessible). Let $\Omega = \Sigma \cup \{x_i\}_{i=1}^{n}$. We recall the following facts about distinguishing tests from [7].

Lemma 1 ([7]). *A complete set of distinguishing tests D for $\mathcal{A}$ can be constructed in time $O(n^5)$. Further, D can be represented as $\binom{n}{2}$ strings in Ω^*, each of length $O(n^2)$.*

3 Conformance Testing

In this section we define the problem of conformance testing *MVPA* and prove some preliminary lemmas.

By "conformance", we mean language equivalence. Given a *specification machine* $\mathcal{S}$ and a "black-box" implementation machine $\mathcal{I}$ that are both deterministic complete modular *MVPA* over signature *Sig*, we want to test if $\mathcal{I}$ is equivalent to $\mathcal{S}$, i.e., whether $L(\mathcal{I}) = L(\mathcal{S})$. We make the following assumptions:

1. $\mathcal{S}$ is minimized and has n states;
2. $\mathcal{I}$ has at most $N = n + \Delta$ states;

Note that assumption 1 is not a restriction since the details of $\mathcal{S}$ are known and hence can be minimized. Assumption 2 is necessary to guarantee that every state of the implementation is explored. Hence, whenever we refer to a specification machine we assume it is minimized. Also, all the automata we refer to from now on are *MVPA*.

A *sample* is a set of well-matched words. Let the length of a sample be the sum of the lengths of the words in the test. A sample T *distinguishes* $\mathcal{I}$ from $\mathcal{S}$, if there is a word $w \in T$ such that w is accepted by exactly one of $\mathcal{S}$ and $\mathcal{I}$. Given a specification machine $\mathcal{S}$ with n states, a Δ-*conformance test* is a sample T of well-matched words that distinguishes every incorrect implementation machine $\mathcal{I}$, that is, $\mathcal{I}$ such that $L(\mathcal{S}) \neq L(\mathcal{I})$, with at most $n + \Delta$ states from $\mathcal{S}$. Given $\mathcal{S}$ and Δ, a *conformance testing algorithm* outputs a Δ-conformance test.

Let us fix a "black box" implementation machine $\mathcal{I}$ with at most $n+\Delta$ states and a specification machine $\mathcal{S}$ with n states such that $\mathcal{I}$ is not equivalent to $\mathcal{S}$. We first focus on the problem of finding a sample T which distinguishes $\mathcal{I}$ from $\mathcal{S}$. Let Q be the accessible states of $\mathcal{I}$, and $\hat{Q}$ the accessible states of $\mathcal{S}$. Let D be a complete set of distinguishing tests for $\mathcal{S}$. Let $(u_{\hat{q_1}\hat{q_2}}, v_{\hat{q_1}\hat{q_2}}) \in D$ be a distinguishing test for $\{\hat{q_1}, \hat{q_2}\}$ and $D_{\hat{q_1}} = \bigcup_{\hat{q_2}} \{(u_{\hat{q_1}\hat{q_2}}, v_{\hat{q_1}\hat{q_2}})\}$.

We find a sample T such that if T does not distinguish an implementation from $\mathcal{S}$ then there exists a homomorphism from the implementation to $\mathcal{S}$. If T does not distinguish $\mathcal{I}$ from $\mathcal{S}$, then Proposition 1 would imply that $L(\mathcal{I}) = L(\mathcal{S})$, contradicting the assumption on $\mathcal{I}$ and $\mathcal{S}$.

We find a set of access strings for the states of $\mathcal{I}$. We then check that the states reached by these strings in $\mathcal{I}$ and $\mathcal{S}$ are indistinguishable with respect to the distinguishing tests. In order to verify that the transitions in $\mathcal{I}$ are correct, we check that the states reached by taking the transitions in both $\mathcal{I}$ and $\mathcal{S}$ are indistinguishable, that is, to verify that a transition from a state q on a symbol a in $\mathcal{I}$ is correct, we check that the states reached in $\mathcal{I}$ and $\mathcal{S}$ on reading ya are indistinguishable, where y is an access string for q in $\mathcal{I}$, and so on.

Let Y be an arbitrary set of access strings for $\mathcal{I}$, and let $Q' = \{state_{\mathcal{I}}(y) \mid y \in Y\}$. For each $q \in Q'$, fix an access string $y_q \in Y$ for q. Let us define a function $h_Y : Q' \to \hat{Q}$ as follows. $h_Y(q) = state_{\mathcal{S}}(y_q)$. We give a characterization of when h_Y is a partial homomorphism and when h_Y is a homomorphism by describing a set of tests.

Definition 2. *A set of access strings Y is called* safe *if it contains $\{\epsilon\} \cup \{(m,p) \mid m \in M \backslash \{m_0\}, p \in P_m\}$ as a subset and satisfies the following conditions:*

*C*1 *For each $y \in Y$, $(m_0, p_0)y \in L(\mathcal{I})$ iff $(m_0, p_0)y \in L(\mathcal{S})$.*
*C*2 *For each $y \in Y$, for each $(u, v) \in D_{state_{\mathcal{S}}(y)}$, $uyv \in L(\mathcal{I})$ iff $uyv \in L(\mathcal{S})$.*
*C*3 *For each $y \in Y$ and $a \in \Sigma^{int}$, for each $(u, v) \in D_{state_{\mathcal{S}}(ya)}$, $uyav \in L(\mathcal{I})$ iff $uyav \in L(\mathcal{S})$.*
*C*4 *For each $y_1, y_2 \in Y$, and for each $(u, v) \in D_{state_{\mathcal{S}}(y_2 y_1 r)}$, $uy_2y_1rv \in L(\mathcal{I})$ iff $uy_2y_1rv \in L(\mathcal{S})$.*

Informally, a safe set of access strings corresponds to a set of states such that the transitions out of these states do not contain any "bad" transitions. Condition $C1$ verifies that an access string reaches a final state of $\mathcal{I}$ iff it reaches a final state of $\mathcal{S}$. Condition $C2$ ensures that, the states reached by a string of the set in the specification and implementation behave similarly. Condition $C3$ ensures that a transition labelled by an internal symbol is not "bad", that is, the states reached in $\mathcal{I}$ and $\mathcal{S}$ after reading the symbol a from states reached by

the same access string exhibit similar behavior. Similarly $C4$ ensures that the return transitions are not "bad". The next lemma states that if Y is a safe set then it defines a partial homomorphism.

Lemma 2. *If Y is safe then h_Y is a partial homomorphism.*

Corollary 1. *If Y is a safe and complete set of access strings for $\mathcal{I}$, then h_Y is a homomorphism.*

If we are given a complete set of access strings Y for $\mathcal{I}$ which contains $\{\epsilon\} \cup \{(m,p) \mid m \in M \backslash \{m_0\}, p \in P_m\}$, then we can use the above characterization to obtain a sample T_Y which distinguishes $\mathcal{I}$ from $\mathcal{S}$. T_Y is the union of the following sets:

- $T_0 = \{(m_0,p_0)y \mid y \in Y\}$.
- $T_1 = \{uyv \mid y \in Y, (u,v) \in D_{state_{\mathcal{S}}(y)}\}$.
- $T_2 = \{uyav \mid y \in Y, (u,v) \in D_{state_{\mathcal{S}}(ya)}\}$.
- $T_3 = \{uy'yrv \mid y,y' \in Y, (u,v) \in D_{state_{\mathcal{S}}(y'y)}\}$.

However, we cannot compute the set of access strings directly, since we do not have knowledge about the internal structure of $\mathcal{I}$. We use the following result from [7]. Let us fix a complete set of access strings $\{x_1, \cdots, x_n\}$ for $\mathcal{S}$ which contains $\{\epsilon\} \cup \{(m,p) \mid m \in M \backslash \{m_0\}, p \in P_m\}$. Let us denote by $x_{\hat{q}}$ the access string for the state $\hat{q} \in \hat{Q}$ in the above set.

Lemma 3 ([7]). *For each $\hat{q} \in \hat{Q}$ and $(u,v) \in D_{\hat{q}}$, let $ux_{\hat{q}}v \in L(\mathcal{I})$ iff $ux_{\hat{q}}v \in L(\mathcal{S})$. Then there exist access strings for the states of $\mathcal{I}$, $y_1, \cdots, y_N$, where $y_i = x_i$ for $1 \leq i \leq n$ and for each $n < i \leq N$, $y_i = y_j a$ or $y_i = y_j y_k r$ for some $a \in \Sigma^{int}$ and $j,k < i$.*

The premise of the above lemma ensures that distinct x_i access distinct states of $\mathcal{I}$. The above lemma states that a complete set of access strings of $\mathcal{I}$ can be represented as a system of $N-n$ equations of the form $y_i = y_j a$ or $y_i = y_j y_k r$. Since $\mathcal{I}$ is given as a "black box", in order to obtain a sample distinguishing $\mathcal{I}$ from $\mathcal{S}$, we need to consider all the $(N|\Sigma| + N^2)^{N-n}$ systems of equations. We denote the set of all systems of equations as Γ.

Definition 3. *Given a complete set of access strings X for the specification $\mathcal{S}$, we denote by $\Gamma(X,\Delta)$, the set of all systems of equations of the form $y_i = ua$ or $y_i = uvr$ for $1 \leq i \leq N-n$ where each of u,v is either an element of X or is y_j for some $j < i$. Given an element $\gamma \in \Gamma(X,\Delta)$, we denote by Y_γ, the set of access strings generated by γ, that is, the elements of X and the word assignments for y_i, $1 \leq i \leq N-n$ which satisfy the equations in γ.*

Next we present the algorithm given in Algorithm 1.1, which takes as input the specification $\mathcal{S}$, a complete set of access strings $X = \{x_1, \cdots, x_n\}$ for $\mathcal{S}$, a complete set of distinguishing tests D for $\mathcal{S}$ and the "black box" implementation

Algorithm 1.1
1 Input: $(\mathcal{S}, X, D, \mathcal{I})$
2 Output: Sample T
3 $T \leftarrow \emptyset$
4 **for** every $\gamma \in \Gamma(X, \Delta)$ **do**
5 $\quad T \leftarrow T \cup T_{Y_\gamma}$
6 **end for**

Algorithm 1.2
1 Input: $(\mathcal{S}, X, D, \mathcal{I})$
2 Output: Sample T
3 $T \leftarrow \emptyset$
4 **for** $l = 1, \cdots, m$ **do**
5 $\quad \gamma \leftarrow Rand(\Gamma(X, \Delta))$
6 $\quad T \leftarrow T \cup T_{Y_\gamma}$
7 **end for**

$\mathcal{I}$, and outputs a sample T which distinguishes $\mathcal{I}$ from $\mathcal{S}$ if $\mathcal{I}$ is an incorrect implementation.

Observe that if $\mathcal{I}$ is equivalent to $\mathcal{S}$, then no sample can distinguish the two. On the other hand, it $\mathcal{I}$ and $\mathcal{S}$ are not equivalent, then the output of Algorithm 1.1, namely T, distinguishes $\mathcal{I}$ from $\mathcal{S}$. In fact T distinguishes any $\mathcal{I}$ which is not equivalent to $\mathcal{S}$, hence the algorithm outputs a Δ-conformance test.

Theorem 2 ([7]). *The length of the Δ-conformance test output by Algorithm 1.1 is $O((a2^\Delta + b)(n + \Delta)dz((n + \Delta)d)^\Delta)$, where a is the maximum length of the strings in X which is $O(2^n)$, b the maximum length of $|u| + |v|$ for any pair $(u, v) \in D$ which is $O(2^n)$, $z = \max_{\hat{q} \in \hat{Q}} |D_{\hat{q}}|$ which is $O(n)$, and $d = (n + \Delta + |\Sigma^{int}|)$.*

For the sake of illustration, we will give a conformance test for the example in Figure 1 for the case with no extra states. That is, let the specification $\mathcal{S}$ be the *MVPA* of Figure 1. Note that it is a minimal *MVPA* (we will exhibit a complete set of distinguishing tests). We give a conformance test which distinguishes every *MVPA* with at most 5 states which is not equivalent to $\mathcal{S}$. First let us fix access strings for every state of $\mathcal{S}$. Let $x_{q_0} = \epsilon$, $x_q = (m, p)br$, $x_{q_1} = (m, p)$, $x_{q_2} = (m, p)a$ and $x_{q_3} = (m, p)b$. Here y_s is an access string for state s, that is, $(m_0, p_0)y_s$ reaches state s. Next let us define a complete set of distinguishing tests. $((m_0, p_0), \epsilon)$ is a distinguishing pair for $\{q_0, q\}$. $((m_0, p_0), ar)$ is a distinguishing pair for $\{q_1, q_2\}$ and $\{q_1, q_3\}$, and $((m_0, p_0), r)$ is a distinguishing pair for $\{q_2, q_3\}$. Since $\Delta = 0$, Γ is a singleton set $\{\gamma\}$ and the corresponding set $Y_\gamma = \{x_{q_0}, x_q, x_{q_1}, x_{q_2}, x_{q_3}\}$. The test T is simply T_{Y_γ} which is the union of the following sets: (The substrings in bold font correspond to the part of the string which comes from the set Y.)

- $T_0 = \{(m_0, p_0), (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{br}, (m_0, p_0)(\mathbf{m}, \mathbf{p}),$
  $(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{a}, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{b}\}$.
- $T_1 = \{(m_0, p_0), (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{br}, (m_0, p_0)(\mathbf{m}, \mathbf{p})ar,$
  $(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{a}ar, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{b}ar, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{a}r,$
  $(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{b}r\}$.
- $T_2 = \{(m_0, p_0)\mathbf{a}, (m_0, p_0)\mathbf{b}, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{bra},$
  $(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{brb}, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{a}ar,$
  $(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{a}r, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{b}ar, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{b}r,$
  $(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{aa}ar, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{aa}r,$

$(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{ab}ar, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{ba}ar,$
$(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{ba}r, (m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{bb}ar,$
$(m_0, p_0)(\mathbf{m}, \mathbf{p})\mathbf{bb}r\}$.

- T_3 has more than 25 strings, hence in interest of space, we will explain how some of the elements of T_3 are constructed. Choose any two strings from Y, say x_{q_1} and x_{q_2}. The state reached on $x_{q_1}x_{q_2}r = (m,p)(m,p)ar$ is q_3. Add the strings $ux_{q_1}x_{q_2}rv$ for every (u,v) which distinguishes q_3 from the other states in the module m, namely, T_3 contains $(m_0, p_0)(\mathbf{m}, \mathbf{p})(\mathbf{m}, \mathbf{p})\mathbf{ar}ar$ and $(m_0, p_0)(\mathbf{m}, \mathbf{p})(\mathbf{m}, \mathbf{p})\mathbf{ar}r$.

4 (R, Δ)-conformance testing

In this section, we consider a relaxed version of the conformance testing problem in which a test is required to distinguish only those implementations which are at a certain distance from the specification. We define the distance of an implementation from a specification to be the number of transitions that need to be changed in the implementation so as to make it equivalent to the specification. The formal definition is given below:

Definition 4. *A MVPA $\mathcal{I}$ is at distance R from $\mathcal{S}$ if the smallest set $D \subseteq Q' \times (\Sigma^{call} \cup \Sigma^{int} \cup Q')$ such that there exists a MVPA $\mathcal{J} = (\{Q''_m\}, \{q''^p_m\}_{p \in P_m}, \delta''_m\}_{m \in M}, F'')$ over Sig satisfying the following conditions has size R.*

- *For each m, $Q''_m = Q'_m$.*
- *$F'' = F'$.*
- *δ'' and δ' differ only on the set D.*

Given a specification *MVPA* $\mathcal{S}$ of size n, an (R, Δ)-*conformance test* is a sample T of well-matched words which distinguishes $\mathcal{I}$ from $\mathcal{S}$, for every $\mathcal{I}$ of size at most $n+\Delta$ which is at distance at least R from $\mathcal{S}$. The test may or may not distinguish implementations whose distance from $\mathcal{S}$ is less than R. An (R, Δ)-*conformance testing algorithm* is an algorithm that takes $\mathcal{S}$, R and Δ as input and outputs an (R, Δ)-conformance test for $\mathcal{S}$. Note that a $(1, \Delta)$-conformance test is the same as a Δ-conformance test.

Next we present a randomized algorithm that outputs an (R, Δ)-test with high probability. We first present a randomized algorithm which distinguishes a particular "black box" implementation $\mathcal{I}$ from a specification $\mathcal{S}$. We use the notation $i \leftarrow Rand(I)$ to denote that i is chosen uniformly at random from the set I.

The randomized algorithm is based on the intuition that if R is large, then there is a large number of equations, that is, a large subset of Γ, such that the sample corresponding to the access strings generated by these equations distinguishes a particular $\mathcal{I}$ from $\mathcal{S}$. Hence if we choose a sample randomly from Γ, then we catch a buggy implementation with some positive probability. In order to obtain a constant probability, the above step is repeated a certain number of times to boost the probability. The algorithm is given in Algorithm 1.2.

For analyzing the algorithm, we need the following lemma relating the distance R between $\mathcal{S}$ and $\mathcal{I}$ to the number of transitions out of a safe set of states. Let $E[Q'']$ denote the set of edges going out of Q'', that is, an edge (q, a) is in $E[Q'']$ if $q \in Q''$, $a \in \Sigma^{int} \cup Q''$ and $\delta(q, a) \notin Q''$.

Lemma 4. *Let $\mathcal{I}$ be at distance at least R from $\mathcal{S}$. Let $Y \supseteq X$ be a set of access strings for $\mathcal{I}$ which is safe. Let Q'' be the states of $\mathcal{I}$ accessed by Y. Then the size of the set $E[Q'']$ is at least R.*

Note that when $L(\mathcal{I}) = L(S)$, no test can distinguish them. So it remains to analyse the probability that the sample output by the algorithm distinguishes $\mathcal{I}$ from $\mathcal{S}$, under the assumption that $\mathcal{I}$ is at distance at least R from $\mathcal{S}$. Let $d = n + \Delta + |\Sigma^{int}|$.

Lemma 5. *Let $\mathcal{I}$ be at distance greater than or equal to $R > 0$ from $\mathcal{S}$. Let $uxv \in L(\mathcal{I})$ iff $uxv \in L(\mathcal{S})$, for every $x \in X$ and $(u, v) \in D_{state_{\mathcal{S}}(x)}$. Probability that for a γ chosen uniformly at random from Γ, Y_γ is an unsafe set, is at least $(\frac{R}{(n+\Delta)d})^{\Delta}$.*

Let $P = (\frac{R}{(n+\Delta)d})^{\Delta}$, where $d = (n + \Delta + |\Sigma^{int}|)$. The next theorem gives the probability that Algorithm 1.2 distinguishes $\mathcal{I}$ from $\mathcal{S}$.

Theorem 3. *Let $\mathcal{I}$ be at distance at least R from $\mathcal{S}$. For any $\epsilon > 0$, the output of Algorithm 1.2 distinguishes $\mathcal{I}$ from $\mathcal{S}$ with probability at least $1 - \epsilon$ after $k = \frac{1}{P}\log(\frac{1}{\epsilon})$ iterations. The length of the sample output by the algorithm is $O((a2^{\Delta} + b)(n + \Delta)dz(\frac{(n+\Delta)d}{R})^{\Delta})$, where a is the maximum length of the strings in X which is $O(2^n)$, b the maximum length of $|u| + |v|$ for any pair $(u, v) \in D$ which is $O(2^n)$, $z = \max_{\hat{q} \in \hat{Q}} |D_{\hat{q}}|$ which is $O(n)$.*

Note that given any $\mathcal{I}$, the output of the algorithm distinguishes $\mathcal{I}$ from $\mathcal{S}$ with high probability. However, it does not guarantee that the output of the algorithm distinguishes every $\mathcal{I}$ from $\mathcal{S}$ with high probability. Next we modify the algorithm by increasing the number of iterations k so that the output of the algorithm distinguishes every $\mathcal{I}$ from $\mathcal{S}$ or in other words is a conformance test. Note that in the case of a deterministic conformance testing algorithm the two are the same, that is, if the output of the algorithm distinguishes $\mathcal{I}$ from $\mathcal{S}$ where $\mathcal{I}$ is unknown, then it is a conformance test, i.e., it distinguishes every $\mathcal{I}$ from $\mathcal{S}$.

Let α be the number of faulty implementations, that is, $\mathcal{I}$ with at most $n+\Delta$ states at distance at least R from $\mathcal{S}$. α is upper bounded by $(n+\Delta)^{(n+\Delta)d}$, total number of implementation machines with at most $n+\Delta$ states. Set $k = \frac{1}{P}\log(\frac{\alpha}{\epsilon})$. Then the output of the algorithm distinguishes a particular $\mathcal{I}$ with probability $1 - \epsilon/\alpha$. So the probability that the output of the algorithm distinguishes every $\mathcal{I}$ is at least $1 - \epsilon$.

Theorem 4. *For any $\epsilon > 0$, the output of Algorithm 1.2 is an (R, Δ)-conformance test with probability at least $1-\epsilon$, when $k = \frac{1}{P}\log(\frac{\alpha}{\epsilon})$. The length of the (R, Δ)-conformance test output by the algorithm is $O((a2^{\Delta} + b)(n + \Delta)^2 d^2 z \log(n +$*

$\Delta)(\frac{(n+\Delta)d}{R})^{\Delta})$, *where* a *is the maximum length of the strings in* X *which is* $O(2^n)$, b *the maximum length of* $|u|+|v|$ *for any pair* $(u,v) \in D$ *which is* $O(2^n)$, $z = \max_{\hat{q}\in\hat{Q}} |D_{\hat{q}}|$ *which is* $O(n)$.

5 Lower bounds for conformance testing

We will first define the specification and implementation machines involved in the proof of the lower bound that we wish to establish, and prove some properties about these automata.

5.1 Specification *MVPA*

Given an $n > 1$ and Σ^{int} containing a, we define an $(n+4)$ state *MVPA* $\mathcal{S}(n, \Sigma^{int})$ over the signature $\langle M, \{P_m\}_{m\in M}, m_0, \hat{\Sigma}\rangle$, where $M = \{m_0, m_1\}$, $P_{m_0} = \{p_0\}, P_{m_1} = \{p_1\}$, and $\hat{\Sigma} = \{(m_0,p_0),(m_1,p_1),r\} \cup \Sigma^{int}$. Figure 2 gives a diagram of $\mathcal{S}(n, \Sigma^{int})$, all transitions not shown are assumed to go to a fail state in the corresponding module. When n and Σ^{int} is clear from the context, we refer to $\mathcal{S}(n, \Sigma^{int})$ as just $\mathcal{S}$.

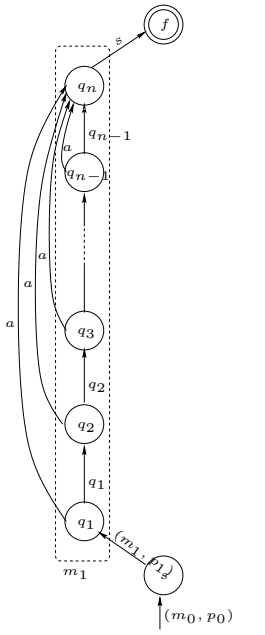

Fig. 2. Specification *MVPA* $\mathcal{S}(n, \Sigma^{int})$

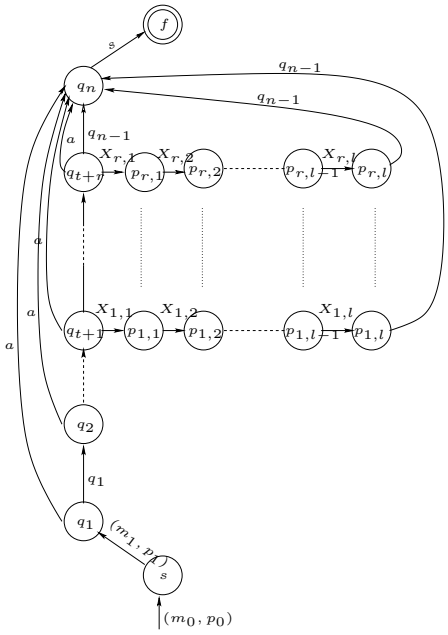

Fig. 3. Implementation *MVPA* $\mathcal{I}(X)$

$\mathcal{S} = (\{Q_m, \{q_m^p\}_{p\in P_m}, \delta_m\}_{m\in M}, F)$ where $Q_{m_0} = \{s, f, d\}$ and $Q_{m_1} = \{q_1, \cdots, q_n, d'\}$. From the start state s, there is a transition on call (m_1, p_1) to q_1 in the

module m_1, and a return transition from q_n in module m_1 to the only accepting state f in module m_0. All the other transition on s and f go to the state d (for dead state). Inside module m_1, there is a return transition $(q_i, q_i, q_{i+1}) \in \delta^{ret}$ for every $1 \leq i < n$. Note that all call transitions on (m_1, p_1) go to q_1. The rest of the transitions in module m_1 go to the dead state d'. From now on we will refer to (m_1, p_1) as c.

Proposition 2 *$L(\mathcal{S}(n, \Sigma^{int})) = \{(m_0, p_0)cw_iar \mid 1 \leq i < n\} \cup \{(m_0, p_0)cw_nr\}$, where w_i is defined inductively as:*

- *$w_1 = \epsilon$, and*
- *$w_i = w_{i-1}cw_{i-1}r$, for $i > 1$.*

5.2 Lower Bound for the (R, Δ)-Conformance Test

In this section, we define the implementation machines and prove the lower bound on the length of the conformance test.

We define a class of implementation machines with $n+4+\Delta$ states which are at distance R from $\mathcal{S}(n, \Sigma^{int})$. Hence these machines take as parameters n, Δ, R and Σ^{int}. Let us fix these parameters for this section. Let us also assume that $\Delta = lR$, where $l \in \mathbb{N}$. We define a template for the implementation machine, which when initialized by appropriate values gives us a class of *MVPA*s. Let $X = \{X_{i,j}\}_{i \in [R], j \in [l]}$ be a set of variables, which are labels of the transitions in the implementation machines we define.

We now define $\mathcal{I}(X)$. $\mathcal{I}(X)$ has the same signature as $\mathcal{S}$ and has all the states of $\mathcal{S}$. In addition, in the module m_1, it has Δ extra states. All the transitions consisting of states common to $\mathcal{S}$ and $\mathcal{I}$ are the same except for those going to the dead states. For each state q_{t+i} where $t = n-(R+1)$ and $i \in [R]$, there is a sequence of l states $p_{i,1}, p_{i,2}, \cdots, p_{i,l}$ such that there is a transition from q_{t+i} to $p_{i,1}$ labelled $X_{i,1}$ and for each $j \in [l-1]$, there is a transition from $p_{i,j}$ to $p_{i,j+1}$ on $X_{i,j+1}$. Finally there is a transition from $p_{i,l}$ to q_n on q_{n-1}. This automaton is shown in Figure 3.

A valuation V for X assigns to every $X_{i,j}$ in X, a symbol from the alphabet of the automaton. A valuation V is *valid* if it satisfies the following constraints:

- If j is even, then $V(X_{i,j}) = p_{i,j-1}$ for every i.
- If j is odd, then there is some $a \in \Sigma^{int}$ such that for every i, $V(X_{i,j}) = a$, or there is some $k \geq 1$ such that $n - kR \in \{\lfloor \frac{n-1}{2} \rfloor, \cdots, n-1\}$ such that for every i, $V(X_{i,j}) = n - kR + (i-1)$, or there is an $\hat{j} < j$ such that for every i, $V(X_{i,j}) = p_{i,\hat{j}}$. Also $X_{i,1} \notin \{a, q_{t+i}\}$.

By $\mathcal{I}(V)$ we mean the implementation machine with the variables in X replaced by the corresponding symbol from the alphabet. We will assume from now on that V is valid.

Next we will prove some properties about $\mathcal{I}$.

Proposition 3 *$\mathcal{I}(V)$ is at distance at least R from $\mathcal{S}$.*

Language of $\mathcal{I}(V)$ Language of $\mathcal{I}(V)$ consists of the words from the language of $\mathcal{S}$ and R new words $u_1, \cdots, u_R$ defined as follows. Let $w_1, \cdots, w_n$ be the unique well-matched words which reach q_i from q_1 given in Proposition 2. Then for every $i \in [R]$ and $j \in [l]$, we define a word $u_{i,j}$ inductively as follows.

- Case $j = 1$:
  - If $V(X_{i,j}) \in \Sigma^{int}$, then $u_{i,j} = w_i V(X_{i,j})$.
  - If $V(X_{i,j}) = q_k$, then $u_{i,j} = w_k c w_i r$.
- Case $j > 1$:
  - If $V(X_{i,j}) \in \Sigma^{int}$, then $u_{i,j} = u_{i,j-1} V(X_{i,j})$.
  - If $V(X_{i,j}) = q_k$, then $u_{i,j} = w_k c u_{i,j-1} r$.
  - If $V(X_{i,j}) = p_{i,\hat{j}}$ for some $\hat{j} < j$, then $u_{i,j} = u_{i,\hat{j}} c u_{i,j-1} r$.

Now we set u_i to be $(m_0, p_0) c w_{n-1} c u_{i,l} r r$.

Proposition 4 *$L(\mathcal{I}(V))$ is a union of $L(\mathcal{S})$ and $\{u_1, \cdots, u_R\}$.*

Proposition 5 *If V_1 and V_2 are two different valid valuations, then $L(\mathcal{I}(V_1)) \cap L(\mathcal{I}(V_2)) = L(\mathcal{S})$. Also no string in $L(\mathcal{I}(V_1)) \backslash L(\mathcal{S})$ is a prefix of a string in $L(\mathcal{I}(V_2)) \backslash L(\mathcal{S})$.*

Proposition 6 $|u_i| \geq 2^{n-R+\frac{\Delta}{2R}-5}$.

Proposition 7 *The number of distinct valid valuations is at least*

$$\prod_{i=1}^{\lfloor \frac{\Delta}{2R} \rfloor - 1} (\lfloor \frac{n-1}{2R} \rfloor + |\Sigma^{int}| + i)$$

Using the above facts, we obtain the following theorem:

Theorem 5. *For every n, Δ, Σ^{int} and $R < n$, there is a specification MVPA $\mathcal{S}$ of size n such that any (R, Δ)-conformance test has at least $\prod_{i=1}^{\lfloor \frac{\Delta}{2R} \rfloor - 1} (\lfloor \frac{n-5}{2R} \rfloor + |\Sigma^{int}| + i)$ strings each of length at least $2^{n-R+\frac{\Delta}{2R}-9}$. Hence the length of the (R, Δ)-conformance test is at least*

$$2^{n-R+\frac{\Delta}{2R}-9} \prod_{i=1}^{\lfloor \frac{\Delta}{2R} \rfloor - 1} (\lfloor \frac{n-5}{2R} \rfloor + |\Sigma^{int}| + i).$$

Discussion. The lower bound as given by Theorem 5 on the size of the (R, Δ)-conformance test is $\Omega((2^{n+\frac{\Delta}{R}-R})(\frac{n}{R} + |\Sigma^{int}| + \frac{\Delta}{R})^{\frac{\Delta}{R}})$ and the upper bound as given by Theorem 4 is $O((2^{n+\Delta})(\frac{n+\Delta+|\Sigma^{int}|}{R})^{\Delta})$. Note that when R is $O(1)$, the upper and the lower bounds match.

6 Conclusions

We investigated the problem of constructing (R, Δ) conformance tests for modular VPAs. We presented a randomized algorithm for constructing such tests, that outputs a test suite which is an (R, Δ) conformance test with high probability. We also presented lower bound proofs that demonstrate that our algorithm is close to optimal. One interesting open problem is to tighten the gap between the lower bound and the upper bound. Another line research would be to explore the connections between (R, Δ) tests and learning [3] and model checking [12, 5].

References

1. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of Recursive State Machines. *ACM Transactions on Programming Languages and Systems*, 27(4):786–818, 2005.
2. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the ACM Symposium on Theory of Computation*, pages 202–211, 2004.
3. T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffan. On the correspondence between conformance testing and regular inference. In *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*, pages 175–189, 2005.
4. A. Friedman and P. Menon. *Fault Detection in Digital Circuits*. Prentice Hall, 1971.
5. A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. *Logic Journal of the IGPL*, 14(5):729–744, 2006.
6. Z. Kohavi. *Switching and Finite Automata Theory*. McGraw Hill, 1978.
7. V. Kumar, P. Madhusudan, and M. Viswanathan. Minimization, learning, and conformance testing of Boolean programs. In *Proceedings of the Internation Conference on Concurrency Theory*, pages 203–217, 2006.
8. V. Kumar and M. Viswanathan. Conformance testing in the presence of multiple faults. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1136–1145, 2005.
9. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines; A survey. *Proceedings of the IEEE*, 84:1090–1126, 1996.
10. R. Linn and M. Üyar, editors. *Conformance Testing methodologies and architechtures for OSI protocols*. IEEE Computer Society Press, 1995.
11. E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies, Annals of Mathematics Studies, Princeton University Press*, 34:129–153, 1956.
12. D. Peled, M. Vardi, and M. Yannakakis. Black Box Checking. *Journal of Automata, Languages, and Combinatorics*, 7(2):225–246, 2002.
13. J. Tretmans. A formal approach to conformance testing. In *Protocol Test Systems*, volume C-19 of *IFIP Transactions*, pages 257–276. 1994.
14. M. Vasilevskii. Fault diagnosis of automata. *Kibernetika*, 4:98–108, 1973.