

# On the Complexity of Error Explanation

Nirman Kumar<sup>1</sup>, Viraj Kumar<sup>2\*</sup>, and Mahesh Viswanathan<sup>2\*\*</sup>

<sup>1</sup> Oracle Corporation, Redwood Shores, CA, USA,  
nirman.kumar@oracle.com

<sup>2</sup> University of Illinois at Urbana-Champaign, Urbana, IL, USA,  
{kumar,vmahesh}@cs.uiuc.edu

**Abstract.** When a system fails to satisfy its specification, the model checker produces an error trace (or counter-example) that demonstrates an undesirable behavior, which is then used in debugging the system. *Error explanation* is the task of discovering errors in the system or the reasons why the system exhibits the error trace. While there has been considerable recent interest in automating this task and developing tools based on different heuristics, there has been very little effort in characterizing the computational complexity of the problem of error explanation. In this paper, we study the complexity of two popular heuristics used in error explanation. The first approach tries to compute the smallest number of system changes that need to be made in order to ensure that the given counter-example is no longer exhibited, with the intuition being that these changes are the errors that need fixing. The second approach relies on the observation that differences between correct and faulty runs of a system shed considerable light on the sources of errors. In this approach, one tries to compute the correct trace of the system that is closest to the counter-example. We consider three commonly used abstractions to model programs and systems, namely, finite state Mealy machines, extended finite state machines and pushdown automata. We show that the first approach of trying to find the fewest program changes is NP-complete no matter which of the three formal models is used to represent the system. Moreover we show that no polynomial factor approximation algorithm for computing the smallest set of changes is possible, unless  $P = NP$ . For the second approach, we present a polynomial time algorithm that finds the closest correct trace, when the program is represented by a Mealy machine or a pushdown automata. When the program is represented by an extended finite state machine, the problem is once again NP-complete, and no polynomial factor approximation algorithm is likely.

## 1 Introduction

Model checking [1] is a popular technique for automated verification of software and hardware systems. One of the principal reasons for its wide spread use is the

---

\* Supported by DARPA/AFOSR MURI award F49620-02-1-0325

\*\* Supported by NSF CCF 04-29639

ability of the model checker to produce a witness to the violation of a property in the form of an error trace (counter-example). While counter-examples are useful in debugging a system, the error traces can be very lengthy and they indicate only the *symptom* of the error. Locating the *cause* of the error (or the bug) is often an onerous task even with a detailed counter-example. Recently, considerable research effort [2–8] has been directed towards automating the process of *error explanation* (or *localizing errors* or *isolating error causes*) to assist in the debugging process by identifying possible causes for the faulty behavior. Error explanation tools are now featured in model checkers such as SLAM [9, 6] and Java PathFinder (JPF) [10, 7].

Error explanation is an intrinsically informal process that admits many heuristic approaches which cannot be justified formally. Most current approaches to this problem rely on two broad philosophical themes for justification. First, in order to explain something (like an error), one has to identify its “causes” [11]. And second is Occam’s principle, which states that a “simpler” explanation is to be always preferred between two competing theories. The different approaches to error explanation primarily differ in what they choose to be the “causal theory” for errors. Two popular heuristics have been widely and successfully used in debugging. The first one relies on the observation that program changes which result in a system that no longer exhibits the offending error trace identify possible causes for the error [12–14, 3, 2]; in accordance with Occam’s principle, one tries to find the minimum number of changes. The second, more popular approach [5–8] relies on the intuition that differences between correct and faulty runs of the system shed considerable light on the sources of errors. This approach tries to find correct runs exhibited by the system that *closely match* the error trace. They then infer the causes of the error from the correct executions and the given error-trace.

While algorithms for these heuristics have been developed based on sophisticated use of SAT solvers and model checkers [12, 13, 3, 2, 5–8], there has been very little effort to study the computational complexity of these methods. In this paper we study the computational complexity of applying the above mentioned heuristics to three commonly used abstractions to model systems<sup>3</sup>. The first and least expressive model we look at is that of Mealy Machines [15], which are finite state machines (FSMs) that produce outputs when transiting from one state to another; the reason we consider Mealy machines and not finite automata is because they are a generalization. The second model we consider are Extended FSMs, which are finite state machines equipped with a finite number of boolean variables which are manipulated in each transition. The third model we examine is that of Pushdown Automata (PDA) which have been widely used

---

<sup>3</sup> We make no effort to judge the practical usefulness of these error explanation approaches. The interested reader is referred to the papers cited here for examples where these heuristics have been effective in debugging.

as a model of software programs in model checking <sup>4</sup>. Once again we consider a generalization of PDAs that produce outputs.

The two approaches to error explanation yield three distinct notions of the ‘smallest distance’ between the given program and a correct program, which we now describe. The precise definitions of these distances depends on the representation used, and will be presented in Section 2.2. We investigate the complexity of computing these distances when the program is abstractly represented by one of the three computation models.

**Minimum edit set** Let  $M$  be an abstract representation of the program. For an input string  $w_i$  and an output string  $w_o$  of the same length as  $w_i$ , an *edit set* is a set of transitions of  $M$  that can be changed so that the resulting program  $M'$  produces output  $w_o$  on input  $w_i$ . A *minimum edit set*  $X_M(w_i, w_o)$  is an edit set of smallest size.

**Closest-output distance** Let  $M$  be an abstract representation of the restriction of the program to correct runs. For an output string  $w_o$ , the *closest-output distance*  $d_M(w_o)$  is the smallest Hamming distance between  $w_o$  and a string  $w$  that can be produced by  $M$ .

**Closest-input distance** Let  $M$  be an abstract representation of the restriction of the program to correct runs. For an input string  $w_i$ , the *closest-input distance*  $d_M(w_i)$  is the smallest Hamming distance between  $w_i$  and a string  $w$  that is in the language of the machine represented by  $M$ .

*Remark 1.* Note that the latter two distances are defined when the *restriction* of the program to correct runs is represented by one of the computation models we consider. The representations we consider are commonly used to model programs, and for most correctness properties of interest (e.g. those expressed in linear temporal logic) the subset of executions of the system satisfying such properties can be expressed using the same kind of abstraction.

*Summary of results* Our results can be summarized as follows. The problem of determining the size of a minimum edit set for given input/output strings and a program represented as a Mealy machine is NP-complete. In addition, there is no polynomial time algorithm for computing an edit set whose size is within a polynomial factor of the size of a minimum edit set unless  $P = NP$ . A couple of points are in order about these results. First, the intractability of this error explanation method for extended finite state machines and PDAs (and boolean programs) follows as a corollary because they are generalizations of the simple Mealy machine model. Second, since we prove these results for a deterministic model, we can also conclude the intractability of this problem for nondeterministic models, which are typically encountered in practice.

---

<sup>4</sup> In software model checking the more commonplace model is actually a boolean program with a stack, which is a combination of extended finite state machines and PDAs. While we do not explicitly consider this model, our results have consequences for doing error explanation for such boolean programs, which we point out.

We provide a more positive answer for the second error explanation approach. When the set of correct executions of a system can be represented by traces of pushdown automata, we present a polynomial time algorithm based on dynamic programming to determine the closest-output distance for a given output string. Since finite state machines are a special case of pushdown automata, this upper bound applies to them as well. However, when the set of correct traces is represented by an extended finite state machine, the results are radically different. Not only is the problem of computing the closest-input distance NP-complete in this case, but we show that it is unlikely that polynomial factor approximation algorithms to compute the closest-input distance in polynomial time exist. Note that the typical model for programs used in model checking is boolean programs, which can be seen as PDAs with boolean variables. Since this model is a generalization of extended finite state machines, our lower bounds imply that the second error explanation method will be intractible for boolean programs as well.

Note also that for the purposes of error explanation, we are not just interested in computing the above distance measures, but rather in computing the closest correct execution. However, the results on computing the above distances have direct consequences to error explanation. The intractibility of computing the closest input distance for extended finite state machines implies that finding the closest correct trace is also intractible. Further, the dynamic programming based algorithm that we present for computing the closest-output distance for PDAs (and FSMs) can be easily modified in a standard manner, to yield not just the distance but also the closest correct trace to the given error trace.

The rest of the paper is organized as follows. In Section 2 we provide the formal definitions of system models and the problems whose complexity we investigate. Section 3 provides the hardness results on computing the minimum edit set. Section 4 provides a polynomial time algorithm to compute the closest output distance and a hardness result for the problem of computing the closest input distance. Finally, we present our conclusions in Section 5.

## 2 Preliminaries

### 2.1 Abstract representations

In this section, we recall the definitions of the various formal models of systems that we consider in this paper, namely, Mealy machines, Extended finite state machines, and Pushdown Automata.

*Mealy Machines* A  $(\Sigma, \Omega)$ -FSM is a deterministic finite state Mealy machine  $M = (V_M, i_M, \delta_M)$  with finite input alphabet  $\Sigma$ , finite output alphabet  $\Omega$ , finite set of states  $V_M$ , initial state  $i_M \in V_M$  and transition function  $\delta_M : V_M \times \Sigma \rightarrow V_M \times \Omega$ . For  $w \in \Sigma^*$ ,  $M(w)$  denotes the string in  $\Omega^*$  generated by  $M$  on input  $w$ . We denote  $\delta_M(u, \sigma) = (v, \omega)$  by the shorthand  $u \xrightarrow{\sigma/\omega}_M v$ .

*Extended FSMs* Suppose  $X$  is a finite set of Boolean variables and  $\mathcal{A}$  is the set of all possible assignments to variables in  $X$ , i.e.  $\mathcal{A} = \{0, 1\}^X$ . Suppose  $\mathcal{B}$  is the set of all finite boolean expressions involving the variables in  $X$  and the constants  $\top$  (true) and  $\perp$  (false). A  $(\Sigma, X)$ -FSM is a tuple  $M = (V_M, s_M, F_M, a_M, g_M, \delta_M)$  where:

1.  $V_M$  is a finite set of states,  $F_M \subseteq V_M$  is the set of final (accepting) states ;
2.  $s_M$  is the initial state,  $a_M \in \mathcal{A}$  is the initial assignment of variables ;
3.  $g_M : V_M \times \Sigma \rightarrow \mathcal{B}$  is the guard function ;
4.  $\delta_M : V_M \times \Sigma \rightarrow V_M \times \mathcal{A} \times \mathcal{A}$  is the transition function.

Executions of a  $(\Sigma, X)$ -FSM are defined as follows. The initial state is  $s_M$  and the initial assignment to the variables is  $a_M$ . If the current state is  $s$ , the current assignment is  $a$ , the input symbol is  $\sigma$  and  $\delta_M(s, \sigma) = (s', a_T, a_F)$ , then

- if  $g_M(s, \sigma) = \perp$ , no transition is possible ;
- if  $g_M(s, \sigma) \neq \perp$ , the next state is  $s'$  and the next assignment is  $a'$ , where  $a' = a_T$  if  $g_M(s, \sigma)$  is satisfied by  $a$ , and  $a' = a_F$  otherwise.

We use the shorthand notation  $(s, \sigma) \xrightarrow[a_F]{a_T} s'$  to represent such a transition. We use the notation  $(s, \sigma) \rightarrow s'$  when the new assignment is identical to the current assignment  $a$ . If  $a \in \mathcal{A}$ , we use the notation  $a[x_i := b]$  to denote the assignment in  $\mathcal{A}$  that is identical to  $a$  except that  $x_i$  is set to  $b$ . We use the notation  $L_M \subseteq \Sigma^*$  to denote the set of strings accepted by  $M$ .

*Remark 2.* Note that any  $(\Sigma, \Omega)$ -FSM  $M = (V_M, i_M, \delta_M)$  can be modeled as a  $(\Sigma \times \Omega, \emptyset)$ -FSM  $M'$  where the guard function for any pair  $(v, (\sigma, \omega))$  is  $\top$  whenever  $M$  produces the output  $\omega$  on input  $\sigma$  from state  $v$ , and  $\perp$  otherwise.

*Pushdown Automata* We formally define Pushdown Automata that produce outputs on each transition. We restrict ourselves slightly to nondeterministic pushdown automata which, on every transition, push or pop at most one symbol from the stack, and consume exactly one input symbol. This model is nevertheless powerful enough to capture visibly pushdown automata [16] or control flow graphs [17], which are typically used to model software systems in the model checking community. A (nondeterministic)  $(\Sigma, \Omega)$ -PDA with finite input alphabet  $\Sigma$  and finite output alphabet  $\Omega$  is a tuple  $M = (V, V_i, \Gamma, \delta)$  where

1.  $V$  is a finite set of states,  $V_i \subseteq V$  is the set of initial states ;
2.  $\Gamma = \Gamma' \cup \{\perp\}$  is a finite stack alphabet,  $\perp$  is the bottom-of-stack symbol ;
3.  $\delta = \delta_c \cup \delta_r \cup \delta_i$  is the transition relation, where  $\delta_c \subseteq (V \times \Sigma \times V \times \Gamma' \times \Omega)$ ,  $\delta_r \subseteq (V \times \Sigma \times \Gamma' \times V \times \Omega)$  and  $\delta_i \subseteq (V \times \Sigma \times V \times \Omega)$ .

A transition  $(u, \sigma, v, \gamma, \omega) \in \delta_c$  is a push-transition where on reading  $\sigma$ ,  $\gamma$  is pushed onto the stack,  $\omega$  is outputted and the state changes from  $u$  to  $v$ . Similarly,  $(u, \sigma, \gamma, v, \omega) \in \delta_r$  is a pop-transition where on reading  $\sigma$ , if  $\gamma \neq \perp$  is the top of the stack, the symbol  $\gamma$  is popped from the top of the stack,  $\omega$  is outputted and the state changes from  $u$  to  $v$ . If the top of the stack is  $\perp$ , no pop-transition

is possible. On internal transitions  $(u, \sigma, v, \omega) \in \delta_i$ , the stack does not change and  $\omega$  is outputted while the state changes from  $u$  to  $v$ . Note that  $(\Sigma, \Omega)$ -PDAs need not be deterministic.

The set of possible stacks  $S$  is  $\Gamma'^* \perp$ . We say that a *run*  $r$  exists from  $(u_1, s_1) \in V \times S$  on input  $w = \sigma_1 \sigma_2 \dots \sigma_k \in \Sigma^*$  if  $\exists (u_1, s_1), (u_2, s_2), \dots, (u_k, s_k)$  such that, for every  $j = 1, \dots, k$ ,  $(u_j, s_j) \in V \times S$  and one of the following transitions  $t_j$  exists in  $\delta$ :

1.  $t_j = (u_j, \sigma_j, u_{j+1}, \gamma, \omega_j) \in \delta_c$  such that  $\gamma \in \Gamma'$  and  $s_{j+1} = \gamma s_j$
2.  $t_j = (u_j, \sigma_j, \gamma, u_{j+1}, \omega_j) \in \delta_r$  such that  $\gamma \in \Gamma'$  and  $s_j = \gamma s_{j+1}$
3.  $t_j = (u_j, \sigma_j, u_{j+1}, \omega_j) \in \delta_i$  and  $s_{j+1} = s_j$ .

In this case, we say that the sequence of transitions  $\bar{t} = t_1 t_2 \dots t_k$  is *consistent* with the run  $r$ .

We say that  $w \in \Sigma^*$  is a *valid input from state*  $v \in V$  if there is a run  $r$  from  $(v, \perp)$  on input  $w$ . We say that  $w \in \Sigma^*$  is a *balanced input from state*  $v \in V$  if there is a run  $r$  from  $(v, \perp)$  on input  $w$  such that for some sequence  $\bar{t}$  of transitions consistent with  $r$ , the number of push-transitions in  $\bar{t}$  is equal to the number of pop-transitions in  $\bar{t}$ . If the output string produced by the sequence  $\bar{t}$  is  $w'$  and the destination of the final transition in  $\bar{t}$  is  $v'$ , we use the notation  $v \xrightarrow{w/w'}_M v'$  to denote the fact that  $M$  produces output  $w'$  on the balanced input  $w$  from  $v$ , and the state changes from  $v$  to  $v'$ .

We say that  $M$  can reach the state  $v$  and can produce the output  $\omega_1 \omega_2 \dots \omega_k$  on input  $w$  if  $w$  is a valid input from some state  $u \in V_i$  and for some sequence of transitions  $t_1 t_2 \dots t_k$  consistent with a run from  $(u, \perp)$  on input  $w$ , the output of  $t_j$  is  $\omega_j$  for every  $1 \leq j \leq k$  and the destination of  $t_k$  is  $v$ . Let  $M(w)$  denote the set of all strings that  $M$  can produce on input  $w$ .

## 2.2 Problem definitions

As mentioned in the introduction, we are interested in the examining the complexity of two heuristics for error explanation. The first heuristic tries to find the smallest number of program transformations that result in the error trace not being exhibited any longer. This problem is related to computing what we call a *minimum edit set* of a program, which we define formally below.

*Minimum edit set* We define a minimum edit set when the program  $M$  is represented as a  $(\Sigma, \Omega)$ -FSM. Given equal length strings  $w_i \in \Sigma^*$ ,  $w_o \in \Omega^*$ , an *edit set* is a set  $X \subseteq V_M \times \Sigma$  such that there is a  $(\Sigma, \Omega)$ -FSM  $M'$  for which  $V_{M'} = V_M$ ,  $i_{M'} = i_M$ ,  $\delta_M$  and  $\delta_{M'}$  differ only on the set  $X$  and  $M'(w_i) = w_o$ . A *minimum edit set*  $X_M(w_i, w_o)$  is a smallest edit set  $X$ .

*Remark 3.* Although we have defined minimum edit set only for Mealy machines, we could easily define it for the other models we consider as well. We shall show that this problem is intractible for Mealy machines. Since the other models are more general than Mealy machines, the intractibility result applies to these other models as well.

The second heuristic tries to find the closest correct computation to the given error trace. This is related to computing the *closest-output distance* that we define below.

*Closest-output distance* We define the closest-output distance when the correct executions of the program are represented as a  $(\Sigma, \Omega)$ -PDA  $M$ . Given a string  $w_o \in \Omega^*$ , the *closest-output distance*  $d_M(w_o)$  is the smallest non-negative integer  $d$  for which there is a string  $w \in M(w_i)$  for some  $w_i \in \Sigma^*$  such that the Hamming-distance between  $w$  and  $w_o$  is  $d$ .

*Remark 4.* We will present a polynomial time algorithm for the problem of computing the closest-output distance, when the correct executions are modeled as a PDA (and a Mealy machine). Therefore, we formally define this problem for a model that has outputs as well, which is more general than a model without outputs. Again we formally define this problem only for PDAs, which is a most general model for which this upper bound applies. Also, for error explanation we would actually be interested in computing the closest correct computation and not just the distance, and we outline how this can be done in Section 4.

Finally, we show that applying the second heuristic when the correct traces are modeled as an extended finite state machine is difficult. We do this by showing that computing another distance measure is difficult; since we are proving a lower bound, this measure is defined for models without outputs.

*Closest-input distance* We define the closest-input distance when the correct executions of the program are represented as a  $(\Sigma, X)$ -FSM  $M$ . Given a string  $w_i \in \Sigma^*$ , the *closest-input distance*  $d_M(w_i)$  is the smallest non-negative integer  $d$  for which there is a string  $w \in L_M$  such that the Hamming-distance between  $w$  and  $w_i$  is  $d$ .

### 3 Computing the Minimum Edit Set

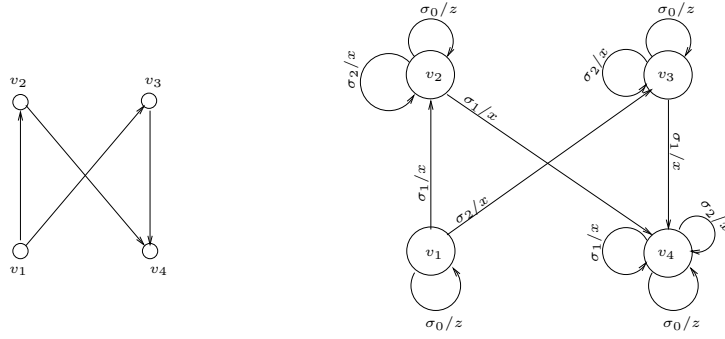
#### 3.1 NP-completeness

Let  $M$  be a  $(\Sigma, \Omega)$ -FSM. We consider the decision version of computing the size of  $X_M(w_i, w_o)$ , i.e. given a non-negative integer  $k$ , decide whether or not  $|X_M(w_i, w_o)| \leq k$ . Clearly, this problem is in NP: we guess an edit set  $X$  of size  $k$  and guess the changes to  $\delta_M$  to be made on the set  $X$ . We then verify if  $M'(w_i) = w_o$  for the resulting  $(\Sigma, \Omega)$ -FSM  $M'$ .

We now show that the decision version of the problem is NP-hard, even when the size of the input and output alphabets ( $\Sigma$  and  $\Omega$ ) are bounded by a constant. We will reduce the HAMILTONIAN-CYCLE problem to our problem. Given a directed graph  $G$ , we construct a  $(\Sigma, \Omega)$ -FSM  $M$  and input/output strings  $w_i$  and  $w_o$  such that any edit set must contain a certain set of transitions of  $M$ . The key idea is to show that this set of transitions is “small” if and only if  $G$  has a Hamiltonian cycle.

*Reduction* The undirected HAMILTONIAN-CYCLE problem for graphs with at most one edge between any pair of vertices and with degree bounded by a constant is NP-complete (see the reduction from 3-CNF in [18]). Since undirected graphs are a special case of directed graphs, HAMILTONIAN-CYCLE is NP-complete for digraphs with outdegree bounded above by a constant  $d$  and with at most one edge between any ordered pair of vertices. Let  $G = (V, E)$  be any such digraph, where  $V = \{v_1, v_2, \dots, v_n\}$  and for every  $i = 1, 2, \dots, n$ , there is a non-negative integer  $m_i \leq d$  and a permutation  $\pi_i$  of  $(1, 2, \dots, n)$  such that  $(v_i, v_{\pi_i(k)}) \in E$  iff  $k \leq m_i$ .

Let  $\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_d\}$  and let  $\Omega = \{x, y, z\}$ . We construct an  $n$ -state  $(\Sigma, \Omega)$ -FSM  $M = (V_M, i_M, \delta_M)$ , where  $V_M = V$ ,  $i_M = v_1$  and for every  $i = 1, \dots, n$ : (1)  $v_i \xrightarrow{\sigma_0/z}_M v_i$ , (2)  $v_i \xrightarrow{\sigma_k/x}_M v_{\pi_i(k)}$  whenever  $1 \leq k \leq m_i$ , and (3)  $v_i \xrightarrow{\sigma_k/x}_M v_i$  whenever  $m_i < k \leq d$ . Figure 1 depicts an example graph and the Mealy machine  $M$  with  $d = 2$ . We use the notation  $\sigma^k$  to denote  $\sigma$  repeated  $k$



**Fig. 1.** A graph  $G$  and the associated FSM  $M$

times. Let  $t_1 = yx^{n-1}$ , and for  $i = 2, \dots, n$  let  $t_i = x^{n-i+1}yx^{i-2}$ . Further, let

$$\begin{aligned} s_{1\dots d} &= (\sigma_0\sigma_1\sigma_0^n)(\sigma_0\sigma_2\sigma_0^n)\dots(\sigma_0\sigma_d\sigma_0^n) \\ w_i &= \sigma_0^{2n} s_{1\dots d} (\sigma_0 s_{1\dots d})^{n-1} \\ w_o &= t_1 t_1 (yxt_1)^d (yxt_2)^d x(xxt_3)^d \dots x(xxt_n)^d \end{aligned}$$

**Lemma 1.**  $|X_M(w_i, w_o)| \leq dn$  iff  $G$  has a Hamiltonian cycle.

*Proof.* Suppose  $G$  has a Hamiltonian cycle  $C$ . We obtain a  $(\Sigma, \Omega)$ -FSM  $M'$  satisfying  $M'(w_i) = w_o$  by modifying  $dn$  transitions of  $M$  as follows:

Since  $C$  is a Hamiltonian cycle in  $G$ , for every  $v_i$  there is exactly one  $v_j$  such that  $(v_j, v_i) \in C$ . For every such  $v_i$ , replace the transition  $v_i \xrightarrow{\sigma_0/z}_M v_i$  with the transition  $v_i \xrightarrow{\sigma_0/y}_{M'} v_j$  (if  $v_i = v_1$ ), and with the transition  $v_i \xrightarrow{\sigma_0/x}_{M'} v_j$  (if  $v_i \neq v_1$ ). This accounts for  $n$  changes.



defined above consists of  $d$  substrings of the form  $\sigma_0\sigma_k\sigma_0^n$ , where  $k = 1, \dots, d$ . Since  $M'(w_i) = w_o$ , it follows that for every  $i = 1, \dots, n$  and every  $k = 1, \dots, d$ ,  $M'$  goes from state  $v_{\pi(i)}$  to  $v_{\pi(i)}$  on input  $\sigma_0\sigma_k$  with output of the form  $tx$  (where  $t \in \{x, y\}$ ). Hence,  $M'$  also satisfies the third property. For the example presented in Figure 1, a solution Mealy machine  $M'$  must be as depicted in Figure 2.

Since  $M'$  satisfies the first and third properties, for every  $i = 1, \dots, n$  and every  $k = 1, \dots, d$ :  $v_{\pi(i+1)} \xrightarrow{\sigma_k/x}_{M'} v_{\pi(i)}$  (where  $\pi(n+1) = \pi(1)$ ). Now  $G = (V, E)$  is not Hamiltonian, so there is at least one  $j \in \{1, \dots, n\}$  such that  $(v_{\pi(j+1)}, v_{\pi(j)}) \notin E$ . Thus, there is no transition of the form  $v_{\pi(j+1)} \xrightarrow{\sigma_k/x}_M v_{\pi(j)}$  in  $M$ . Also, for every  $i \neq j$ , there is at most one transition of the form  $v_{\pi(i+1)} \xrightarrow{\sigma_k/x}_M v_{\pi(i)}$  in  $M$ . Hence,  $M'$  differs from  $M$  in at least  $d + (d-1)(n-1)$  transitions.

Furthermore, since  $M'$  satisfies the second property, it clearly differs from  $M$  on the  $n$   $\sigma_0$ -labeled transitions. Thus,  $M'$  differs from  $M$  in at least  $dn + 1$  transitions. Hence,  $|X_M(w_i, w_o)| > dn$ .  $\square$

This completes the proof of NP-completeness for the decision version of computing  $|X_M(w_i, w_o)|$  when  $M$  is a  $(\Sigma, \Omega)$ -FSM.

*Remark 5.* Note that  $(\Sigma, \Omega)$ -FSMs are restricted versions of  $(\Sigma \times \Omega, X)$ -FSMs (as observed in Remark 2) and  $(\Sigma, \Omega)$ -PDAs. It is easy to show that the decision version of computing  $|X_M(w_i, w_o)|$  is also in NP when  $M$  is a  $(\Sigma \times \Omega, X)$ -FSM or a  $(\Sigma, \Omega)$ -PDA. Hence, the decision version of computing  $|X_M(w_i, w_o)|$  is also NP-complete for these models.

### 3.2 Inapproximability Result

Given a  $(\Sigma, \Omega)$ -FSM  $M = (V_M, i_M, \delta_M)$  and equal length strings  $w_i \in \Sigma^*$  and  $w_o \in \Omega^*$ , we prove that if the minimum edit set has size  $d$ , then for every positive constant  $k$  there is no polynomial time algorithm to construct a  $(\Sigma, \Omega)$ -FSM  $M' = (V_M, i_M, \delta_{M'})$  such that  $M'(w_i) = w_o$  and  $\delta_M$  and  $\delta_{M'}$  differ on a set of size at most  $d^k$ , unless  $P = NP$ .

Our proof is in three steps. We first prove a variant of a result by Pitt and Warmuth [19]: Given a positive integer  $k$  and a set of input/output pairs of strings  $P$  for which the smallest FSM consistent with  $P$  (i.e. an FSM which produces output  $w'$  on input  $w$  for every pair  $(w, w') \in P$ ) has  $n$  states, there is no efficient algorithm to construct an FSM consistent with  $P$  having at most  $n^k$  states (unless  $P = NP$ ). Next, given such a set of pairs  $P$ , we carefully construct an FSM  $M$  and a single input/output pair  $(w_i, w_o)$  such that the minimum edit set  $X_M(w_i, w_o)$  has size  $\Theta(n)$ . Finally, we show that *any* edit set  $X$  can be efficiently modified to yield an FSM with  $|X|$  states that is consistent with  $P$ . We put these three results together to complete our proof.

**Definition 1.** Given a finite set  $\Sigma$  and two finite sets  $POS, NEG \subseteq \Sigma^*$ , a deterministic finite automata (DFA) is said to be consistent with  $(POS, NEG)$  if it accepts all strings in  $POS$  and rejects all strings in  $NEG$ .

Under the assumption that  $P \neq NP$ , Pitt and Warmuth [19] prove the following inapproximability result for the minimum consistent DFA problem.

**Theorem 1 (Pitt-Warmuth).** *Given a finite set  $\Sigma$  such that  $|\Sigma| \geq 2$ , and two finite sets of strings  $POS, NEG \subseteq \Sigma^*$ , for any positive integer  $k$  there is no polynomial time algorithm to find a DFA with at most  $n^k$  states, where  $n$  is the number of states in the smallest DFA that is consistent with  $(POS, NEG)$ .*

Using the result above we prove the following:

**Lemma 2.** *Given a finite set of input/output pairs  $P$  in  $\Sigma^* \times \Omega^*$ , let  $n$  be the minimum number such that there is a  $(\Sigma, \Omega)$ -FSM  $M$  with  $n$  states that is consistent with  $P$  (i.e. for every pair  $(w_i, w_o) \in P$ ,  $M(w_i) = w_o$ ). Then, assuming  $P \neq NP$ , for any positive constant  $k$ , there is no polynomial-time algorithm to find a consistent  $(\Sigma, \Omega)$ -FSM  $M'$  with at most  $n^k$  states. This result holds even if  $|\Sigma|$  and  $|\Omega|$  are bounded by suitable constants.*

*Proof.* We reduce the minimum consistent DFA problem to the above problem. Consider an instance of the minimum consistent DFA problem with input alphabet  $\Sigma$  and sets  $POS, NEG \subseteq \Sigma^*$ . Let  $\Sigma' = \Sigma \cup \{c\}$  where  $c \notin \Sigma$  and  $\Omega = \{0, 1, 2\}$ . Consider the set of input output pairs  $P = P^+ \cup P^-$ , where

$$P^+ = \{(w_i, w_o) \mid w_i = xc, x \in POS \text{ and } w_o = 2^{|w_i|}1\}$$

$$P^- = \{(w_i, w_o) \mid w_i = xc, x \in NEG \text{ and } w_o = 2^{|w_i|}0\}$$

If there is a  $(\Sigma', \Omega)$ -FSM  $M$  with  $n$  states consistent with the set of pairs in  $P$  then we can construct a DFA  $M'$  with  $n$  states where we neglect the output symbols and label states accept or reject by the output produced on the input  $c$  from that state. Clearly this DFA is consistent with  $(POS, NEG)$ . Conversely, if there is a DFA consistent with  $(POS, NEG)$  then we can produce a  $(\Sigma', \Omega)$ -FSM  $M$  that is consistent with  $P$  as follows: the output on all inputs in  $\Sigma$  is 2, and the transition from every state on input  $c$  leads to the same state, with output 1 if the state is an accepting state of the DFA and with output 0 otherwise. Clearly the FSM produced is consistent with  $P$ .  $\square$

For the rest of this section, we fix an input alphabet  $\Sigma$  (with  $|\Sigma| = c$ , a constant), a finite output alphabet  $\Omega$ , and a finite set  $P = \{(w_{i1}, w_{o1}), \dots, (w_{ik}, w_{ok})\}$  of input/output pairs of strings over  $\Sigma^* \times \Omega^*$ , with  $|w_{is}| = |w_{os}|$  for  $1 \leq s \leq k$ . A necessary and sufficient condition for the existence of a  $(\Sigma, \Omega)$ -FSM consistent with the pairs in  $P$  is the following: For any  $w \in \Sigma^*$ , if  $w$  is a prefix of both  $w_{ip}$  and  $w_{iq}$  then the prefixes of length  $|w|$  of  $w_{op}$  and  $w_{oq}$  must be identical. If this condition is satisfied, it is easy to construct a  $(\Sigma, \Omega)$ -FSM consistent with  $P$  which has at most  $m = (\sum_{s=1}^k |w_{is}|) + 1$  states. As a corollary, if the smallest  $(\Sigma, \Omega)$ -FSM consistent with  $P$  has  $n$  states, then  $n \leq m$ .

Let  $\sigma_0, \sigma_1 \notin \Sigma$  and let  $x, y, z \notin \Omega$ . Let  $\Sigma' = \Sigma \cup \{\sigma_0, \sigma_1\}$  and  $\Omega' = \Omega \cup \{x, y, z\}$ . We construct a  $(\Sigma', \Omega')$ -FSM  $M = (V_M, v_1, \delta_M)$  with  $m$  states such that for every  $v \in V_M$ ,  $v \xrightarrow{\sigma_1/x} v_1$  and for every  $\sigma \in \Sigma$ ,  $v \xrightarrow{\sigma/z} v$ ; and further, for some fixed permutation  $\pi$  of  $(1, 2, \dots, m)$  such that  $\pi(1) = 1$ ,

$$v_{\pi(1)} \xrightarrow{\sigma_0/y} v_{\pi(2)} \xrightarrow{\sigma_0/x} v_{\pi(3)} \xrightarrow{\sigma_0/x} \dots \xrightarrow{\sigma_0/x} v_{\pi(n)} \xrightarrow{\sigma_0/x} v_{\pi(1)}$$

i.e. the transitions on input  $\sigma_0$  form a Hamiltonian cycle, with output  $x$  for every transition other than the transition from the initial state  $v_1 = v_{\pi(1)}$ , for which the output is  $y$ . Let  $t_1 = yx^{m-1}$  and for every  $i = 2, \dots, m$  let  $t_i = x^{m-i+1}yx^{i-2}$ . Consider

$$w_i = \sigma_0^{2m}[(\sigma_0\sigma_1\sigma_0^m)(\sigma_0^2\sigma_1\sigma_0^m)\dots(\sigma_0^{m-1}\sigma_1\sigma_0^m)][(w_{i1}\sigma_1)(w_{i2}\sigma_1)\dots(w_{ik}\sigma_1)]$$

$$w_o = t_1t_1[(yxt_1)(yxx_1)\dots(yx^{m-2}xt_1)][(w_{o1}x)(w_{o2}x)\dots(w_{ok}x)]$$

We claim that  $|X_M(w_i, w_o)|$  is  $\Theta(n)$ , where  $n$  is the number of states in the smallest  $(\Sigma, \Omega)$ -FSM consistent with  $P$ . The proof of this claim follows from the following two lemmas:

**Lemma 3.**  $|X_M(w_i, w_o)| \leq cn$ .

*Proof.* Notice that  $\sigma_0^{2m}$  is a prefix of  $w_i$  and the corresponding prefix of  $w_o$  is  $t_1t_1 = yx^{m-1}yx^{m-1}$ . By an argument similar to the one used in the reduction of Subsection 3.1, we can prove that any  $(\Sigma', \Omega')$ -FSM  $M' = (V_M, v_1, \delta_{M'})$  such that  $M'(w_i) = w_o$  must have the structure of a Hamiltonian cycle on the input  $\sigma_0$ , where the output on  $\sigma_0$  is  $y$  from the initial state  $v_1$ , and  $x$  from every other state; furthermore, all transitions of  $M'$  on input  $\sigma_1$  from any state must go to the initial state  $v_1$  with output  $x$ . Notice that these properties are true of  $M$ . Hence,  $M$  only needs to be modified so that it produces output  $w_{os}$  on input  $w_{is}$  for each  $1 \leq s \leq k$ .

As remarked earlier,  $m \geq n$ . Since there is an  $n$ -state  $(\Sigma, \Omega)$ -FSM consistent with  $P$ , it is possible to select an  $n$ -state subset  $V$  of  $V_M$  and modify only the  $\Sigma$ -transitions from states in  $V$  to obtain a  $(\Sigma', \Omega')$ -FSM  $M'$  such that  $M'(w_{is}) = w_{os}$  for every  $1 \leq s \leq k$ . It now immediately follows that  $M'(w_i) = w_o$ . Hence,  $|X_M(w_i, w_o)| \leq |\Sigma|n = cn$ .  $\square$

**Lemma 4.**  $|X_M(w_i, w_o)| \geq n$ .

*Proof.* As remarked earlier,  $m \geq n$ . Let  $|X_M(w_i, w_o)| = d$  and suppose we have made  $d$  changes to  $M$ , yielding  $M'$  such that  $M'(w_i) = w_o$  and hence, for every  $1 \leq s \leq k$ ,  $M'(w_{is}) = w_{os}$ . Thus, there are at most  $d$  states  $v$  for which at least one transition of the form  $v \xrightarrow{\sigma/z}_M v$  ( $\sigma \in \Sigma$ ) has been changed. Hence, there are at least  $m - d$  states in  $M'$  such that  $v \xrightarrow{\sigma/z}_{M'} v$  for every  $\sigma \in \Sigma$ . We claim that we can discard all such states  $v$  and modify the resulting  $(\Sigma', \Omega')$ -FSM to obtain a  $(\Sigma, \Omega)$ -FSM consistent with  $P$ .

Consider any state  $v$  such that  $v \xrightarrow{\sigma/z}_{M'} v$  for every  $\sigma \in \Sigma$ . Thus on an input  $w_{is}$ ,  $1 \leq s \leq k$ , if we can ever reach  $v$ , it must be the last state, since the output from  $v$  on any input symbol in  $\Sigma$  is  $z$  and  $w_{os}$  does not contain  $z$ . Clearly  $v$  cannot be the start state of  $M'$ . So  $v$  can be discarded from  $M'$  and it is still possible to construct a  $(\Sigma', \Omega')$ -FSM  $M''$  such that  $M''(w_{is}) = w_{os}$  for every  $1 \leq s \leq k$  (all edges into  $v$  can be sent to some other state).

This process can be repeated for all such states  $v$ , resulting in a  $(\Sigma', \Omega')$ -FSM  $\hat{M}$  with at most  $d$  states such that  $\hat{M}(w_{is}) = w_{os}$  for every  $1 \leq s \leq k$ . Now, by discarding all  $\sigma_0$  and  $\sigma_1$  transitions from  $\hat{M}$ , we obtain a  $(\Sigma, \Omega)$ -FSM consistent with  $P$  which has  $d = |X_M(w_i, w_o)|$  states. Hence,  $|X_M(w_i, w_o)| \geq n$ .  $\square$

Recalling that  $c = |\Sigma|$  is a constant, we conclude that  $|X_M(w_i, w_o)|$  is  $\Theta(n)$ .

Suppose there is a positive integer  $k$  and a polynomial time algorithm to compute a  $(\Sigma', \Omega')$ -FSM  $M'$  that differs from  $M$  in at most  $|X_M(w_i, w_o)|^k$  transitions for which  $M'(w_i) = w_o$ . Using an argument similar to the one in Lemma 4, we can modify  $M'$  in polynomial time by discarding at least  $m - |X_M(w_i, w_o)|^k$  states and the  $\sigma_0$  and  $\sigma_1$  transitions to obtain a  $\Theta(n^k)$ -state  $(\Sigma, \Omega)$ -FSM consistent with  $P$ . By Lemma 2, this would imply that  $P = NP$ . Thus we have the following

**Theorem 2.** *Given a  $(\Sigma, \Omega)$ -FSM  $M$  and equal length strings  $w_i \in \Sigma^*$  and  $w_o \in \Omega^*$ , for any positive integer  $k$  there is no polynomial time algorithm to compute an edit set of size  $|X_M(w_i, w_o)|^k$  unless  $P = NP$ .*

*Remark 6.* By observing that  $(\Sigma, \Omega)$ -FSMs are restricted versions of  $(\Sigma \times \Omega, X)$ -FSMs and  $(\Sigma, \Omega)$ -PDAs, we obtain similar inapproximability results when the program is represented using either of these abstractions.

## 4 Computing the Closest-output distance

### 4.1 Upper bound for FSMs and PDAs

We will prove a polynomial upper bound on the time to compute the closest-output distance when the correct executions of a program are represented as a  $(\Sigma, \Omega)$ -PDA. The closest-output distance can be expressed using two simple recurrences, and our algorithm uses a straightforward dynamic programming approach to solve these recurrences.

Let  $w_o \in \Omega^*$  and let  $M = (V, V_i, \Gamma, \delta)$  be a  $(\Sigma, \Omega)$ -PDA. We consider the problem of computing  $d_M(w_o)$ . Let the length of  $w_o$ , denoted by  $|w_o|$ , be  $L$ . For every  $1 \leq i \leq j \leq L$  let  $w_o(i, j)$  denote the substring of  $w_o$  from the  $i$ -th to the  $j$ -th position and let  $w_o(i)$  denote the  $i$ -th letter of  $w_o$  (i.e.  $w_o(i) = w_o(i, i)$ ).

For every  $v \in V$  and every  $1 \leq i \leq L$ , let  $P(v, i)$  denote the Hamming distance of the closest string to  $w_o(i, L)$  that can be produced by  $M$  on a some valid input  $w$  starting from state  $v$ . Also, for every  $u, v \in V$  and every  $1 \leq i \leq j \leq L$ , let  $B(u, v, i, j)$  denote the Hamming distance of the closest string to  $w_o(i, j)$  that can be produced by  $M$  on some balanced input  $w$  such that the state changes from  $u$  to  $v$ . By definition,  $d_M(w_o) = \min_{v_i \in V_i} P(v_i, 1)$ .

Let  $[\omega_1 \neq \omega_2]$  be 1 if  $\omega_1 \neq \omega_2$  and 0 otherwise. For notational convenience, let  $P(v, i) = 0$  if  $i > L$  and let  $B(u, v, i + 1, i) = 0$  if  $u = v$ , and  $B(u, v, i + 1, i) = \infty$  otherwise. We observe that if  $w$  is a balanced string from state  $u$ , then  $w$  must be of one of the following two forms:

1.  $w = \sigma_i w_1$  where  $(u, \sigma_i, v_1, \omega) \in \delta_i$  for some  $v_1 \in V$  and  $\omega \in \Omega$ , and  $w_1$  is a balanced string from  $v_1$ ; or
2.  $w = \sigma_c w_1 \sigma_r w_2$  where  $(u, \sigma_c, v_1, \gamma, \omega) \in \delta_c$ ,  $v_1 \xrightarrow{w_1/t}_M v_2$  and  $(v_2, \sigma_r, \gamma, v_3, \omega') \in \delta_r$  for some  $v_1, v_2, v_3 \in V$ ,  $\gamma \in \Gamma$ ,  $\omega, \omega' \in \Omega$  and  $t \in \Omega^*$ , and  $w_1$  is a balanced string from  $v_1$  and  $w_2$  is a balanced string from  $v_3$ .

Note that  $w$  can be of the latter form only if  $|w| \geq 2$ . Thus, for every  $1 \leq i \leq j \leq L$ ,  $B(u, v, i, j) = b_1$  if  $j - i < 2$  and  $B(u, v, i, j) = \min(b_1, b_2)$  otherwise, where

$$b_1 = \min[\omega \neq w_o(i)] + B(v_1, v, i + 1, j)$$

minimum over all  $\sigma, u_1, \omega$  such that  $(u, \sigma, v_1, \omega) \in \delta_i$

$$b_2 = \min[\omega_c \neq w_o(i)] + B(v_1, v_2, i + 1, k) + [\omega_r \neq w_o(k + 1)] + B(v_3, v, k + 2, j)$$

minimum over all  $\sigma_c, v_1, v_2, \sigma_r, v_3, \gamma, k$  such that  $i < k < j$  and  $(u, \sigma_c, v_1, \gamma, \omega_c) \in \delta_c$ ,  $(v_2, \sigma_r, \gamma, \omega_r) \in \delta_r$  for some  $\sigma_c, \sigma_r \in \Sigma, \gamma \in \Gamma', v_1, v_2, v_3 \in V$

We also observe that any valid input  $w$  from state  $u$  must be of the following three forms:

1.  $w = \sigma_c w_1$  where  $(u, \sigma_c, v_1, \gamma, \omega) \in \delta_c$  for some  $\sigma_c \in \Sigma, v_1 \in V, \gamma \in \Gamma', \omega \in \Omega$ , and  $w_1$  is a valid input from  $v_1$ ; or
2.  $w = w_1 w_2$  where  $w_1$  is a balanced string from  $u$ ,  $u \xrightarrow{w_1/t} v_1$  for some  $t \in \Omega^*$  and  $v_1 \in V$ , and  $w_2$  is a valid input from  $v_1$ ; or
3.  $w$  is a balanced string from  $u$ .

Thus, for every  $1 \leq i \leq L$ ,  $P(v, i) = \min(p_1, p_2, p_3)$  where

$$p_1 = \min_{\sigma_c \in \Sigma, v_1 \in V, \gamma \in \Gamma', \omega_c \in \Omega} \{[\omega_c \neq w_o(i)] + P(v_1, i + 1) \mid (u, \sigma_c, v_1, \gamma, \omega_c) \in \delta_c\}$$

$$p_2 = \min_{i \leq k < L, v_1 \in V} B(v, v_1, i, k) + P(v_1, k + 1)$$

$$p_3 = \min_{v_1 \in V} B(v, v_1, i, L)$$

The required value  $\min_{v_i \in V_i} P(v_i, 1)$  can easily be computed using dynamic programming in  $O(|\Sigma|^2 \cdot |\Gamma| \cdot |V|^5 L^3)$  time, which is polynomial in the size of the input. By observing that a  $(\Sigma, \Omega)$ -FSM is a special case of a  $(\Sigma, \Omega)$ -PDA, we obtain the following

**Theorem 3.** *There is a polynomial time algorithm for computing the closest-output distance when the correct executions of the program are represented as a  $(\Sigma, \Omega)$ -PDA or a  $(\Sigma, \Omega)$ -FSM.*

*Remark 7.* Note that the polynomial-time dynamic programming algorithm to compute  $d_M(w_o)$  can easily be modified to compute a string  $w \in \Sigma^*$  such that the Hamming distance between  $w_o$  and some string  $w' \in M(w)$  is  $d_M(w_o)$ .

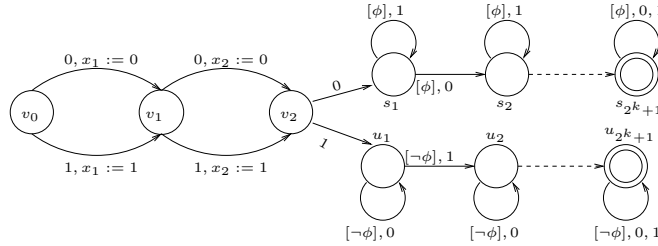
*Remark 8.* Most programs are infinite-state systems and need to be abstracted to form PDAs (or extended finite state machines); consequently these abstractions describe more than just the correct executions of the program. Hence, the input string computed by the above dynamic programming algorithm may not be *legal*, i.e. it may not correspond to a correct execution of the program. While we do not know of a general technique to compute the closest-output distance when inputs are constrained to be legal, the following technique can be used in practice: Using the procedure by Lawler [20], the above dynamic programming algorithm can be used to compute input strings corresponding to the closest-output distance, the second-closest-output distance,  $\dots$ , the  $k^{th}$ -closest-output distance, in time polynomial in  $k$  and the size of the input. These input strings can be examined in order, and the first legal input string can then be chosen.

## 4.2 Hardness results for Extended FSM representation

Given a  $(\Sigma, X)$ -FSM  $M$  and a string  $w_i \in \Sigma^*$ , we show that the decision version of computing  $d_M(w_i)$  is NP-complete and further, there is no polynomial-time algorithm to compute  $d_M(w_i)$  to within any given polynomial factor (unless  $P = NP$ ). Briefly, given a SAT formula  $\phi$  over a set  $X$  of  $n$  variables, we construct a  $(\Sigma, X)$ -FSM  $M$  and a string  $w_i \in \Sigma^*$  such that every string  $w \in L_M$  identifies an assignment  $a_w$  of the variables in  $X$ , and if  $w$  is “close” to  $w_i$ , then  $a_w$  satisfies  $\phi$ . The results follow from the NP-completeness of SAT.

Let  $\Sigma = \{0, 1\}$ , let  $X$  be a set of  $n$  boolean variables, and let  $\phi$  be any SAT formula over variables in  $X$ . For every positive integer  $k$ , let  $N(k) = n^k + 1$  and construct the following  $(\Sigma, X)$ -FSM  $M_k = (V, v_0, F, a_0, g, \delta)$ :

1.  $V = \{v_0, v_1, \dots, v_n\} \cup \{u_1, \dots, u_{N(k)}\} \cup \{s_1, \dots, s_{N(k)}\}$ ,  $F = \{s_{N(k)}, u_{N(k)}\}$ ;
2.  $g(v_i, j) = \top$  for  $i = 0, \dots, n$  and  $j = 0, 1$ ;
3.  $g(u_i, j) = \neg\phi$  and  $g(s_i, j) = \phi$  for  $i = 1, \dots, N(k)$  and  $j = 0, 1$ ;
4.  $(v_{i-1}, 0) \xrightarrow[x_i:=0]{x_i:=0} v_i$  and  $(v_{i-1}, 1) \xrightarrow[x_i:=1]{x_i:=1} v_i$  for  $i = 1, \dots, n$ ;
5.  $(v_n, 0) \rightarrow s_1$  and  $(v_n, 1) \rightarrow u_1$ ;
6.  $(s_i, 0) \rightarrow s_{i+1}$ ,  $(s_i, 1) \rightarrow s_i$ ,  $(u_i, 0) \rightarrow u_i$ ,  $(u_i, 1) \rightarrow u_{i+1}$  for  $1 \leq i \leq N(k) - 1$ ;
7.  $(s_{N(k)}, j) \rightarrow s_{N(k)}$  and  $(u_{N(k)}, j) \rightarrow u_{N(k)}$  for  $j = 0, 1$ ;
8. the initial state is  $v_0$ , and the initial assignment  $a_0$  sets all variables to 0.



**Fig. 3.** The FSM for  $n = 2$

Figure 3 shows the FSM for  $n = 2$  variables. The transition arrows are labeled by the input symbol 0 or 1. The change in assignment (if any) is given after the input symbol while the enabling condition is given before the input symbol if it is not always true.

Note that the size of  $M_k$  is polynomial in the size of the inputs. Let  $w_i = 0^{n+N(k)}$ . By the construction of  $M_k$ , every  $w' \in L_{M_k}$  of length  $n + N(k)$  is either of the form  $w0^{N(k)}$  or  $w1^{N(k)}$  where  $w \in \Sigma^n$ . Note that  $w$  uniquely determines an assignment  $a_w$  of the variables of  $X$ . It is immediately clear from the construction of  $M_k$  that the Hamming distance between  $w_i$  and  $w'$  is at most  $n$  if  $a_w$  satisfies  $\phi$ , and is at least  $N(k)$  otherwise.

The decision problem  $d_{M_k}(w_i) \leq n$  is clearly in NP: we guess a string  $w'$  of length equal to  $w_i$  and verify in polynomial time if  $w' \in L_{M_k}$  and the Hamming distance between  $w'$  and  $w_i$  is at most  $n$ . If so, then as argued above, the prefix  $w$  of length  $n$  of  $w'$  uniquely determines a satisfying assignment to the SAT formula  $\phi$ . Since  $\phi$  was an arbitrary SAT formula, we conclude that this decision problem is NP-complete.

We now show that, unless  $P = NP$ , there is no polynomial time algorithm for computing  $d_{M_k}(w_i)$  to within a polynomial approximation factor. Suppose, for the sake of contradiction, that such an algorithm  $A$  exists. Let  $A(M_k, w_i)$  denote the output of  $A$  for the input  $M_k$  and  $w_i$  defined above. Since  $d_{M_k}(w_i) \leq n$  iff  $\phi$  is satisfiable, it follows that  $A(M_k, w_i) < N(k)$  iff  $\phi$  is satisfiable. Thus, no such polynomial-time algorithm  $A$  exists unless  $SAT \in P$ . Hence, we have the following

**Theorem 4.** *Let  $M$  be a  $(\Sigma, X)$ -FSM and let  $w_i \in \Sigma^*$  such that  $d_M(w_i) = d$ . For any positive integer  $k$ , there is no polynomial time algorithm to decide whether or not  $d_M(w_i) \leq d^k$ , unless  $P = NP$ .*

*Remark 9.* The typical model for programs used in model checking is boolean programs, which can be seen as PDAs with boolean variables. Since this model is a generalization of extended finite state machines, our hardness result applies to boolean programs as well.

*Remark 10.* As mentioned in Remark 8, programs may be abstracted as extended finite state machines, and thereby describe more than just the correct executions of the program. The above hardness result for extended FSMs clearly extends to this more general case as well.

## 5 Conclusions

We have proved upper and lower bounds for two popular heuristics used in automated error explanation for various models encountered in formal verification. Based on our observations, we can draw two important conclusions. First, our lower bounds provide justification for algorithms based on SAT solvers that have been proposed in the literature. These algorithms are likely to be the most efficient that one can hope to design. Second, since the problem of determining the minimum edit set is intractible even for Mealy machines, it is unlikely that error explanation tools based on this heuristic will scale up to large software programs. On the other hand, the closest correct trace to a counter-example can be computed efficiently for PDAs (and hence for finite state models as well). The intractibility of this problem for extended finite state machines is a consequence of the well-known state space explosion problem, and does not seem intrinsic to the heuristic itself.

## References

1. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (2000)

2. Zeller, A.: Isolating cause-effect chains for computer programs. In: Proceedings of the ACM Symposium on the Foundations of Software Engineering. (2002) 1–10
3. Zeller, A., Hildebrandt, R.: Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering* **28** (2002) 183–200
4. Jin, H., Ravi, K., Somenzi, F.: Fate and free will in error traces. In: Proceedings of Conference on Tools and Algorithms for Construction and Analysis of Systems. Volume 2031 of Lecture Notes in Computer Science., Springer (2002) 445–459
5. Renieris, M., Reiss, S.: Fault localization with nearest neighbor queries. In: Proceedings of the Conference on Automated Software Engineering. (2003)
6. Ball, T., Naik, M., Rajamani, S.: From symptom to cause: Localizing errors in counterexample traces. In: Proceedings of the ACM Symposium on the Principles of Programming Languages. (2003) 97–105
7. Groce, A., Visser, W.: What went wrong: Explaining counterexamples. In: Proceedings of the SPIN Workshop on Model Checking of Software. (2003) 121–135
8. Groce, A.: Error explanation with distance metrics. In: Proceedings of Conference on Tools and Algorithms for Construction and Analysis of Systems. (2004) 108–122
9. Ball, T., Rajamani, S.K.: The SLAM project: Debugging system software via static analysis. In: Proceedings of the ACM Symposium on the Principles of Programming Languages. (2002) 1–3
10. Brat, G., Havelund, K., Park, S., Visser, W.: Java PathFinder – A second generation of a Java model checker. In: Proceedings of the Workshop on Advances in Verification. (2000)
11. Lewis, D.: Causation. *Journal of Philosophy* **70** (1973) 556–567
12. Zeller, A.: Yesterday, my program worked. Today, it does not. Why? In: Proceedings of the ACM Symposium on the Foundations of Software Engineering. (1999) 253–267
13. Tip, F., Dinesh, T.B.: A slicing-based approach for locating type errors. *ACM Transactions on Software Engineering and Methodology* **10** (2001) 5–55
14. Bhargavan, K., Gunter, C.A., Kim, M., Lee, I., Obradovic, D., Sokolsky, O., Viswanathan, M.: Verisim: Formal analysis of network simulations. *IEEE Transactions on Software Engineering* **28** (2002) 129–145
15. Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation*. Addison Wesley (1979)
16. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proceedings of the ACM Symposium on the Theory of Computation. (2004)
17. Reps, T., Horwitz, S., Sagiv, M.: Precise interprocedural dataflow analysis via graph reachability. In: Proceedings of the ACM Symposium on the Principles of Programming Languages. (1995) 49–61
18. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*. McGraw-Hill Higher Education (2001)
19. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM* **40** (1993) 95–142
20. Lawler, E.L.: A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science* **18** (1972) 401–405