Completeness of FSM Test Suites Reconsidered

No Author Given

No Institute Given

Abstract. A fault domain that has been widely studied in black-box conformance testing is the class of finite state machines (FSMs) with at most k extra states. Numerous methods for generating test suites have been proposed that guarantee fault coverage for this class. These test suites grow exponentially in k, so one can only run them for small k. But the assumption that k is small is not realistic in practice. As a result, completeness for this fault domain has limited practical significance. As an alternative, we propose (much larger) fault domains that capture the assumption that when bugs in an implementation introduce extra states, these states can be reached via a few (at most k) transitions from states reachable via a set A of common scenarios. Preliminary evidence suggests these fault domains, which contain FSMs with an exponential number of extra states (in k), are of practical use for testing network protocols. We present a sufficient condition for *k-A-completeness* of test suites with respect to these fault domains, phrased entirely in terms of properties of their testing tree. Our condition implies k-A-completeness of two prominent test suite generation algorithms, the Wp and HSI methods. Counterexamples show that three other approaches, the H, SPY and SPYH methods, do not always generate k-A-complete test suites.

Keywords: conformance testing · finite state machines · Mealy machines · apartness · observation tree · k-A-complete test suites

1 Introduction

We revisit the classic problem of black-box conformance testing [23] in a simple setting in which both specifications and implementations can be described as (deterministic, complete) finite state machines (FSMs), a.k.a. Mealy machines. Ideally, given a specification FSM $\mathcal{S}$, a tester would like to have a finite set of tests T that is complete in the sense that an implementation FSM $\mathcal{M}$ will pass all tests in T if and only if $\mathcal{M}$ is equivalent to S. Unfortunately, such a test suite does not exist: if N is the number of inputs in the longest test in T then an implementation $\mathcal{M}$ may behave like $\mathcal{S}$ for the first N inputs, but differently from that point onwards. Even though $\mathcal{M}$ is not equivalent to $\mathcal{S}$, it will pass all tests in T. This motivates the use of a *fault domain*, a collection of FSMs that reflects the tester's assumptions about faults that may occur in an implementation and that need to be detected during testing.

A fault domain that has been widely studied is the class of finite state machines (FSMs) with at most m states. Test suites that detect any fault in this class are called m-complete. The idea of m-complete test suites can be traced back to Moore [27] and Hennie [17]. Numerous methods for constructing m-complete test suites have been proposed, for different types of transition system models, see for instance [42,4,44,14,32,30,23,6,31,35,36,37,2,26,21,40,15]. We refer to [23,5,26] for overviews and further references. The interest in m-complete test suites is somewhat surprising, given that in a black-box setting there is typically no sensible way to bound the number of possible states of an implementation to a small m. After all, each additional Boolean variable in an implementation potentially doubles the number of states. This is problematic, since the size of m-complete test suites grows exponentially in $m - n$, where n is the number of states of the specification. Actually, Moore [27] was just describing gedanken-experiments and not aiming for practical methods. He introduced the example of combination lock machines, and was therefore well aware of the combinatorial explosions in m-complete test suites. Hennie [17] also observed the exponential blow-up in m-complete test suites and wrote "Further work is needed before it will be practical to design checking experiments in which the number of states may increase appreciably." In two classic papers, Vasilevskii [42] and Chow [4] independently showed that m-complete test suites can be constructed with a size that is polynomial in the size of the specification FSM, for fixed value of $k = m - n$. Nevertheless, also the test suites generated by their W-method grow exponentially in k. Interestingly, Chow [4] argued that it is not unreasonable to assume a valid bound on the number of states in the implementation "if we have spent adequate effort in analyzing the specification and constructing the design". Thus Chow actually assumed a white-box setting in order to justify his method.

Active automata learning, a.k.a. model learning, is emerging as a highly effective bug-funding technique with many applications [1,29,39,18]. For instance, using model learning, Fiterău-Broştean e.a. [13,10,12,8] found numerous serious security vulnerabilities and non-conformance issues in implementations of several major network and security protocols including DTLS, SSH and TCP. Black-box conformance testing, which is used to find counterexamples for hypothesis models, has become the main bottleneck in applications of model learning [39,43]. This provides motivation and urgency to come up with fault domains that provide a better fit with applications and allow us to come up with smaller test suites [21].

In response to these challenges, the contributions of this article are as follows:

1. As an alternative, we propose fault domains $\mathcal{U}_k^A$ that contain all FSMs in which any state can be reached by first performing a sequence from some set A (typically a state cover for the specification), followed by k arbitrary inputs, for some small k. These fault domains contain FSMs with a number of extra states that grows exponentially in k. Analysis of a large collection of FSM models of implementations of the DTLS, TLS, SSH and EDHOC protocols provides evidence that the new fault domains $\mathcal{U}_k^A$ make sense from a practical perspective.
2. Based on ideas from [6,40,41], we present a sufficient condition for *k-A-completeness* of test suites with respect to these fault domains, phrased

entirely in terms of properties of their testing tree. We present a $\Theta(N^2)$-time algorithm to check this condition for a testing tree with N states.

3. We show that our sufficient condition implies k-A-completeness of two prominent approaches for test suite generation: the Wp-method of Fujiwara et al [14], and the HSI-method of Luo et al [24] and Petrenko et al [44,32]. The W-method of Vasilevskii [42] and Chow [4], and the UIOv-method of Chan et al [3] are instances of the Wp-method, and the ADS-method of Lee and Yannakakis [22] and the hybrid ADS method of Smeenk et al [36] are instances of the HSI-method. This means that, indirectly, k-A-completeness of these methods follows as well. Hence these m-complete test suite generation methods are complete for much larger fault domains than the ones for which they were designed originally. These larger fault domains can be of practical interest, even for small values of $k = m - n$.
4. We present counterexamples showing that three other prominent test generation methods, the H-method of Dorofeeva et al [6], the SPY-method of Simão, Petrenko and Yevtushenko [35] and the SPYH-method of Soucha and Bogdanov [37], do not always generate k-A-complete test suites.

The rest of this article is structured as follows. First we recall some basic definitions in Section 2. Section 3 introduces k-A-complete test suites, and shows how they strengthen the notion of m-completeness. Next, we present our sufficient condition for k-A-completeness in Section 4. Based on this condition, Section 5 establishes k-A-completeness of the Wp and HSI methods. Finally, Section 6, discusses implications of our results and directions for future research. Proofs, counterexamples and experimental results are deferred to appendices.

2 Preliminaries

In this section, we recall a number of key concepts that play a role in this article: partial functions, sequences, Mealy machines, observation trees, and test suites.

2.1 Partial Functions and Sequences

We write $f \colon X \rightharpoonup Y$ to denote that f is a partial function from X to Y and write $f(x)\downarrow$ to mean that f is defined on x, that is, $\exists y \in Y \colon f(x) = y$, and conversely write $f(x)\uparrow$ if f is undefined for x. Often, we identify a partial function $f \colon X \rightharpoonup Y$ with the set $\{(x, y) \in X \times Y \mid f(x) = y\}$. We use Kleene equality on partial functions, which states that on a given argument either both functions are undefined, or both are defined and their values on that argument are equal.

Throughout this paper, we fix a nonempty, finite set I of *inputs* and a set O of *outputs*. We use standard notations for sequences. If X is a set then X^* denotes the set of finite *sequences* (also called *words*) over X. For k a natural number, $X^{\leq k}$ denotes the set of sequences over X with length at most k. We write ϵ to denote the empty sequence, X^+ for the set $X^* \setminus \{\epsilon\}$, x to denote the sequence consisting of a single element $x \in X$, and $\sigma \cdot \rho$ (or simply $\sigma\rho$) to denote the concatenation of two sequences $\sigma, \rho \in X^*$. The concatenation operation is extended to sets of sequences by pointwise extension. We write $|\sigma|$ to denote the length of σ. For a sequence $\tau = \rho\,\sigma$ we say that ρ and σ are a prefix and a suffix of τ, respectively. We write $\rho \leq \tau$ iff ρ is a prefix of τ. A set $W \subseteq X^*$ is *prefix-closed* if any prefix of a word in W is also in W, that is, for all $\rho, \tau \in X^*$ with $\rho \leq \tau$, $\tau \in W$ implies $\rho \in W$. For $W \subseteq X^*$, $\mathit{Pref}(W)$ denotes the *prefix-closure* of W, that is, the set $\{\rho \in X^* \mid \exists \tau \in W : \rho \leq \tau\}$ of all prefixes of elements of W. If $\sigma = x\rho$ is a word over X with $x \in X$, then we write $\mathsf{hd}(\sigma)$ for x, and $\mathsf{tl}(\sigma)$ for ρ.

2.2 Mealy machines

In this subsection, we recall the definition of Finite State Machines (FSMs) or, as we will call them, Mealy machines.

Definition 2.1 (Mealy machine). *A* Mealy machine *is a tuple $\mathcal{M} = (Q, q_0, \delta, \lambda)$, where Q is a finite set of* states*, $q_0 \in Q$ is the* initial state*, $\delta\colon Q \times I \rightharpoonup Q$ is a (partial)* transition function*, and $\lambda\colon Q \times I \rightharpoonup O$ is a (partial)* output function *that satisfies $\lambda(q,i){\downarrow} \Leftrightarrow \delta(q,i){\downarrow}$, for $q \in Q$ and $i \in I$. We use superscript $\mathcal{M}$ to disambiguate to which Mealy machine we refer, e.g. $Q^{\mathcal{M}}$, $q_0^{\mathcal{M}}$, $\delta^{\mathcal{M}}$ and $\lambda^{\mathcal{M}}$. We write $q \xrightarrow{i/o} q'$, for $q, q' \in Q$, $i \in I$, $o \in O$ to denote $\lambda(q,i) = o$ and $\delta(q,i) = q'$. We call a state $q \in Q$* complete *iff q has an outgoing transition for each input, that is, $\delta(q,i){\downarrow}$, for all $i \in I$. A set of states $W \subseteq Q$ is* complete *iff each state in W is complete. The Mealy machine $\mathcal{M}$ is* complete *iff Q is complete.*

The transition and output functions are lifted to sequences in the usual way. Let $q, q' \in Q$, $\sigma \in I^*$ and $\rho \in O^*$. We write $q \xrightarrow{\sigma/\rho} q'$ to denote $\lambda(q,\sigma) = \rho$ and $\delta(q,\sigma) = q'$. We write $q \xrightarrow{\sigma/\rho}$ if there is a $q' \in Q$ with $q \xrightarrow{\sigma/\rho} q'$, we write $q \xrightarrow{\sigma} q'$ if there is a $\rho \in O^*$ with $q \xrightarrow{\sigma/\rho} q'$, and we write $q \xrightarrow{+} q'$ if there is a $\sigma \in I^+$ with $q \xrightarrow{\sigma} q'$. If $q_0 \xrightarrow{\sigma} q$ then we say that q is reachable *via σ.*

A state cover *for $\mathcal{M}$ is a finite, prefix-closed set of input sequences $A \subset I^*$ such that, for every $q \in Q$, there is a $\sigma \in A$ such that q is reachable via σ. A state cover A is* minimal *if each state of $\mathcal{M}$ is reached by exactly one sequence from A. We say that $\mathcal{M}$ is* initially connected *if it has a state cover. We will only consider Mealy machines that are initially connected.*

Definition 2.2 (Semantics and minimality). *The* semantics *of a state q of a Mealy machine $\mathcal{M}$ is the map $[\![q]\!]^{\mathcal{M}}\colon I^* \rightharpoonup O^*$ defined by $[\![q]\!]^{\mathcal{M}}(\sigma) = \lambda^{\mathcal{M}}(q,\sigma)$.*

States q, r of Mealy machines $\mathcal{M}$ and $\mathcal{N}$, respectively, are equivalent*, written $q \approx r$, iff $[\![q]\!]^{\mathcal{M}} = [\![r]\!]^{\mathcal{N}}$. Mealy machines $\mathcal{M}$ and $\mathcal{N}$ are* equivalent*, written $\mathcal{M} \approx \mathcal{N}$, iff their initial states are equivalent: $q_0^{\mathcal{M}} \approx q_0^{\mathcal{N}}$. A Mealy machine $\mathcal{M}$ is* minimal *iff, for all pairs of states q, q', $q \approx q'$ iff $q = q'$.*

Example 2.3. Figure 1 shows an example (taken from [26]) with a graphical representation of two minimal, complete Mealy machines that are inequivalent, since the input sequence *aba* triggers different output sequences in both machines.

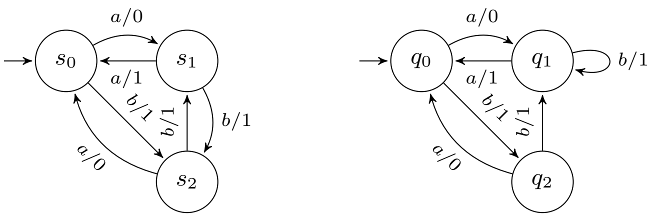

Fig. 1: A specification $\mathcal{S}$ (left) and an inequivalent implementation $\mathcal{M}$ (right).

2.3 Observation Trees

A *functional simulation* is a function between Mealy machines that preserves the initial state and the transition and output functions.

Definition 2.4 (Simulation). *A* functional simulation *between Mealy machines $\mathcal{M}$ and $\mathcal{N}$ is a function $f\colon Q^{\mathcal{M}} \to Q^{\mathcal{N}}$ satisfying $f(q_0^{\mathcal{M}}) = q_0^{\mathcal{N}}$ and*

$$\delta^{\mathcal{M}}(q,i)\downarrow \quad \Rightarrow \quad f(\delta^{\mathcal{M}}(q,i)) = \delta^{\mathcal{N}}(f(q),i) \text{ and } \lambda^{\mathcal{M}}(q,i) = \lambda^{\mathcal{N}}(f(q),i).$$

We write $f\colon \mathcal{M} \to \mathcal{N}$ if f is a functional simulation between $\mathcal{M}$ and $\mathcal{N}$.

Note that if $f\colon \mathcal{M} \to \mathcal{N}$, each i-transition from a state q of $\mathcal{M}$ can be matched by an i-transition from the state $f(q)$ of $\mathcal{N}$.

For a given Mealy machine $\mathcal{M}$, an *observation tree* for $\mathcal{M}$ is a Mealy machine itself that represents the inputs and outputs that have been observed during testing of $\mathcal{M}$. Using functional simulations, we define it formally as follows.

Definition 2.5 (Observation tree). *A Mealy machine $\mathcal{T}$ is a* tree *iff for each $q \in Q^{\mathcal{T}}$ there is a unique $\sigma \in I^*$ s.t. q is reachable via σ. We write $\mathsf{access}(q)$ for the sequence of inputs leading to q. For $U \subseteq Q^{\mathcal{T}}$, we define $\mathsf{access}(U) = \{\mathsf{access}(q) \mid q \in U\}$. For $q \neq q_0^{\mathcal{T}}$, we write $\mathsf{parent}(q)$ for the unique state q' with an outgoing transition to q. A tree $\mathcal{T}$ is an* observation tree *for a Mealy machine $\mathcal{M}$ iff there is a functional simulation f from $\mathcal{T}$ to $\mathcal{M}$.*

Example 2.6. Figure 2(right) shows an observation tree $\mathcal{T}$ for the Mealy machine $\mathcal{S}$ of Figure 2(left). Mealy machine $\mathcal{S}$, an example taken from [37], models the behavior of a turnstile. Initially, the turnstile is locked (L), but when a coin is inserted (c) then, although no response is observed (N), the machine becomes unlocked (U). When a user pushes the bar (p) in the initial state, the turnstile is locked (L), but when the bar is pushed in the unlocked state it is free (F) and the user may pass. State colors indicates the functional simulation from $\mathcal{T}$ to $\mathcal{S}$.

2.4 Test Suites

We recall some basic vocabulary of conformance testing for Mealy machines.

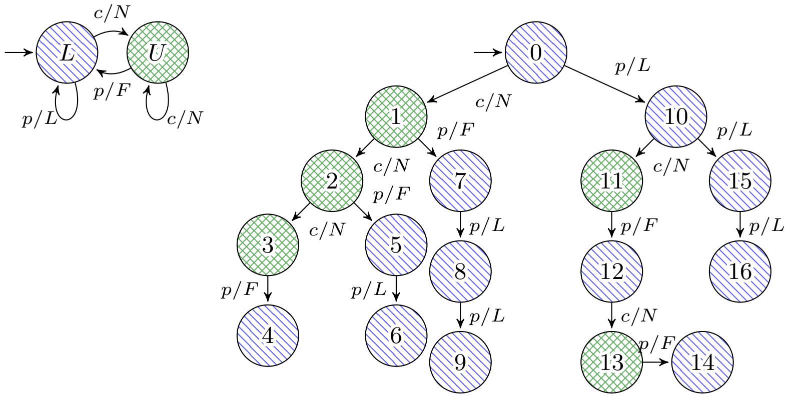

Fig. 2: A Mealy machine $\mathcal{S}$ (left) and an observation tree $\mathcal{T}$ for $\mathcal{S}$ (right).

Definition 2.7 (Test suites). *Let $\mathcal{S}$ be a Mealy machine. A sequence $\sigma \in I^*$ with $\delta^{\mathcal{S}}(q_0, \sigma)\downarrow$ is called a* test case *(or simply a* test*) for $\mathcal{S}$. A* test suite *T for $\mathcal{S}$ is a finite set of tests for $\mathcal{S}$. A Mealy machine $\mathcal{M}$* passes *test σ for $\mathcal{S}$ iff $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) = \lambda^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$, and passes test suite T for $\mathcal{S}$ iff it passes all tests in T.*

Observe that when $\mathcal{M}$ passes a test σ for $\mathcal{S}$ it also passes all proper prefixes of σ. This means that only the maximal tests from a test suite T (tests that are not a proper prefix of another test in T) need to be executed to determine whether $\mathcal{M}$ passes T. Also note that when $\mathcal{M}$ passes a test σ, we may conclude $\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma)\downarrow$.

We like to think of test suites as observation trees. Thus, for instance, the test suite $T = \{cccp, ccpp, cppp, pcpcp, ppp\}$ for the Mealy machine $\mathcal{S}$ of Figure 2(left) corresponds to the observation tree of Figure 2(right). The definition below describes the general procedure for constructing a testing tree for a given test suite T for a specification $\mathcal{S}$. The states of the testing tree are simply all the prefixes of tests in T. Since T may be empty but a tree needs to have at least one state, we require that the empty sequence ϵ is a state.

Definition 2.8 (Testing tree). *Suppose T is a test suite for a Mealy machine $\mathcal{S}$. Then the* testing tree $\mathsf{Tree}(\mathcal{S}, T)$ *is the observation tree $\mathcal{T}$ given by:*

- *$Q^{\mathcal{T}} = \{\epsilon\} \cup Pref(T)$ and $q_0^{\mathcal{T}} = \epsilon$,*
- *For all $\sigma \in I^*$ and $i \in I$ with $\sigma i \in Q^{\mathcal{T}}$, $\delta^{\mathcal{T}}(\sigma, i) = \sigma i$,*
- *For all $\sigma \in I^*$ and $i \in I$ with $\sigma i \in Q^{\mathcal{T}}$, $\lambda^{\mathcal{T}}(\sigma, i) = \lambda^{\mathcal{S}}(\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma), i)$.*

There is a functional simulation from a testing tree to the specification that was used during its construction.

Lemma 2.9. *The function f that maps each state σ of $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$ to the state $\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$ of $\mathcal{S}$ is a functional simulation.*

The next lemma, which follows from the definitions, illustrates the usefulness of testing trees: a Mealy machine $\mathcal{M}$ passes a test suite T for $\mathcal{S}$ iff there exists a functional simulation from $\mathsf{Tree}(\mathcal{S}, T)$ to $\mathcal{M}$.

Lemma 2.10. *Suppose $\mathcal{S}$ and $\mathcal{M}$ are Mealy machines, T is a test suite for $\mathcal{S}$, and $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$. Suppose function f maps each state σ of $\mathcal{T}$ to state $\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma)$ of $\mathcal{M}$. Then $f \colon \mathcal{T} \to \mathcal{M}$ iff $\mathcal{M}$ passes T.*

3 Fault Domains and Test Suite Completeness

For any (finite) test suite T for $\mathcal{S}$, we can trivially construct a faulty implementation that passes T. This justifies the following definition of a *fault domain*, which reflects the tester's assumptions about faults that may occur in an implementation and that need to be detected during testing.

Definition 3.1 (Fault domains and $\mathcal{U}$-completeness). *Let $\mathcal{S}$ be a Mealy machine. A* fault domain *is a set $\mathcal{U}$ of Mealy machines. A test suite T for $\mathcal{S}$ is* $\mathcal{U}$-complete *if, for each $\mathcal{M} \in \mathcal{U}$, $\mathcal{M}$ passes T implies $\mathcal{M} \approx \mathcal{S}$.*

A particular class of fault domains that has been widely studied is based on the maximal number of states that implementations may have.

Definition 3.2. *Let $m > 0$. Then $\mathcal{U}_m$ is the set of all Mealy machines with at most m states.*

In the literature, $\mathcal{U}_m$-complete test suites are usually called *m-complete*. Suppose $m \geq n$, where n is the number of states of a specification $\mathcal{S}$. Given that the size of m-complete test suites grows exponentially in $m - n$, a tester cannot possibly consider all Mealy machines with up to m states, if $m - n$ is large. Therefore, we will propose alternative and much larger fault domains that can still be fully explored during testing.

In applications, a state cover A typically contains the shortest sequences to reach states of specification $\mathcal{S}$, corresponding to common scenarios ("happy flow") to reach those states. We consider fault domains of Mealy machines in which all states can be reached by first performing an access sequence from A and then k arbitrary inputs, for some small k. These fault domains capture the tester's assumption that when bugs in the implementation introduce extra states, these extra states can be reached via a few transitions from states reachable via some common scenarios. The next definition formally introduces the corresponding fault domains $\mathcal{U}_k^A$. The notion of $\mathcal{U}_k^A$-completeness was previously proposed by Maarse [25], in his thesis on query learning using action refinement.

Definition 3.3. *Let k be a natural number and let $A \subseteq I^*$. Then $\mathcal{U}_k^A$ is the set of all Mealy machines $\mathcal{M}$ such that every state of $\mathcal{M}$ can be reached by an input sequence $\sigma\rho$, for some $\sigma \in A$ and $\rho \in I^{\leq k}$.*

Example 3.4. Mealy machine $\mathcal{M}$ from Figure 3 is contained in fault domain $\mathcal{U}_1^A$, for $A = \{\epsilon, c\}$, since all states of $\mathcal{M}$ can be reached via at most one transition from the states L' and U' that are reachable via A.

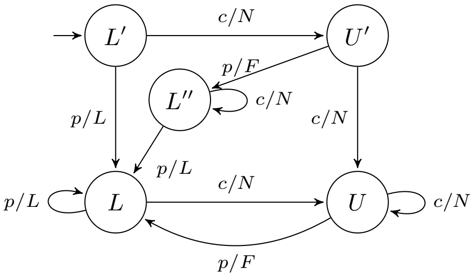

Fig. 3: A Mealy machine $\mathcal{M}$ contained in fault domain $\mathcal{U}_1^A$, with $A = \{\epsilon, c\}$.

Eccentricity. The definition of $\mathcal{U}_k^A$ is closely related to the fundamental concept of *eccentricity* known from graph theory [7]. Consider a directed graph $G = (V, E)$. For vertices $w, v \in V$, let $d(w, v)$ be the length of a shortest path from w to v, or ∞ if no such a path exists. The *eccentricity* $\epsilon(w)$ of a vertex $w \in V$ is defined as the maximum distance from w to any other vertex:

$$\epsilon(w) = \max_{v \in V} d(w, v).$$

This definition generalizes naturally to subsets of vertices. The distance from a set $W \subseteq V$ of vertices to a vertex $v \in V$, is the length of a shortest path from some vertex in W to v, or ∞ if no such path exists.

$$d(W, v) = \min_{w \in W} d(w, v)$$

For a set W of vertices, the *eccentricity* $\epsilon(W)$ is then defined as the maximum distance from W to any other vertex:

$$\epsilon(W) = \max_{v \in V} d(W, v).$$

We view Mealy machines as directed graphs in the obvious way. In the example of Figure 3, the eccentricity of initial state L' is 2 (since every state of $\mathcal{M}$ can be reached with at most 2 transitions from L') and the eccentricity of state U' is ∞ (since there is no path from U' to L'). The eccentricity of $\{L', U'\}$ is 1, since state L can be reached with a single transition from L', and states L'' and U can be reached with a single transition from U'.

Fault domain $\mathcal{U}_k^A$ can alternatively be defined as the set of Mealy machines $\mathcal{M}$ for which the eccentricity of the set of states reachable via A is at most k. Note that, for a set of vertices W, $\epsilon(W)$ can be computed in linear time by contracting all elements of W to a single vertex w, followed by a breadth-first search from w.

Number of extra states. The Mealy machine of Figure 3, which is contained in fault domain $\mathcal{U}_1^A$, has two states (L' and U') that are reached via a sequence from A, and three extra states that can be reached via a single transition from these two states. More generally, if A is a prefix closed set with n sequences and the set of inputs I contains l elements, then at most n states can be reached via sequence from A, and at most $nl - n + 1$ additional states can be reached via a single transition from states already reached by A. A second step from A may lead to $l(nl - n + 1)$ extra states, etc. This means that, for $k > 0$, the fault domain $\mathcal{U}_k^A$ contains Mealy machines with up to $(\sum_{j=0}^{k-1} l^j)(nl - n + 1) + n$ states. This number grows exponentially in k when there are at least 2 inputs. Even for small values of k, the number of states may increase appreciably. Consider, for instance, the Mealy machine model for the Free BSD 10.2 TCP server that was obtained through black-box learning by Fiterău-Broştean et al [11]. This model has 55 states and 13 inputs, so if A is a minimal state cover, then fault domain $\mathcal{U}_2^A$ contains Mealy machines with up to 9309 states.

Even though the size of the Mealy machines in $\mathcal{U}_k^A$ grows exponentially in k, fault domain $\mathcal{U}_m$ is not contained in fault domain $\mathcal{U}_k^A$ if $m = |A| + k$. For instance, the machine of Figure 2 has two states and is therefore contained in $\mathcal{U}_2$. However, this machine is not contained in $\mathcal{U}_0^A$ if we take $A = \{\epsilon, p\}$. We will need to extend $\mathcal{U}_k^A$ in order to obtain a proper inclusion. Suppose that A is a minimal state cover for a minimal specification $\mathcal{S}$. Then states in $\mathcal{S}$ reached by sequences from A will be pairwise inequivalent. Methods for generating m-complete test suites typically first check whether states of implementation $\mathcal{M}$ reached by sequences from A are also inequivalent. This means that these methods exclude any model $\mathcal{M}$ in which two distinct sequences from A reach equivalent states of $\mathcal{M}$. This motivates the following definition:

Definition 3.5. *Let $A \subseteq I^*$. Then $\mathcal{U}^A$ is the set of all Mealy machines $\mathcal{M}$ such that there are $\sigma, \rho \in A$ with $\sigma \neq \rho$ and $\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) \approx \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \rho)$.*

Note that $\mathcal{U}^A$ is infinite and contains Mealy machines with arbitrarily many states. Under reasonable assumptions, we can show that fault domain $\mathcal{U}_m$ is contained in fault domain $\mathcal{U}_k^A \cup \mathcal{U}^A$.

Theorem 3.6. *Let $A \subset I^*$ be a finite set of input sequences with $\epsilon \in A$. Let k and m be natural numbers with $m = |A| + k$. Then $\mathcal{U}_m \subseteq \mathcal{U}_k^A \cup \mathcal{U}^A$.*

The converse inclusion of Theorem 3.6 does not hold, as $\mathcal{U}^A$ may contain Mealy machines with an unbounded number of states. However, the following inclusion holds trivially when $k = 0$.

Proposition 3.7. *Let $A \subset I^*$ be a finite set of input sequences with $|A| = m$. Then $\mathcal{U}_0^A \subseteq \mathcal{U}_m$ (and thus $\mathcal{U}_0^A \cup \mathcal{U}^A \subseteq \mathcal{U}_m \cup \mathcal{U}^A$).*

We refer to $\mathcal{U}_k^A \cup \mathcal{U}^A$-complete test suites as *k-A-complete*. By Theorem 3.6, any k-A-complete test suite is also m-complete, if $m = |A| + k$. Below we give an example to show that, for $k > 0$, m-complete test suites generated by the SPYH-method [37] are in general not k-A-complete, if $m = |A| + k$. Appendix D contains variations of this example, which demonstrate that the SPY-method [35] and the H-method [6] are not k-A-complete either.

Example 3.8. Consider specification $\mathcal{S}$ and testing tree from Figure 2. This specification and the corresponding test suite $T = \{cccp, ccpp, cppp, pcpcp, ppp\}$ were both taken from [37], where the SPYH-method was used to generate T, which was shown to be 3-complete for $\mathcal{S}$. Consider the minimal state cover $A = \{\epsilon, c\}$ for $\mathcal{S}$. Mealy machine $\mathcal{M}$ from Figure 3 is contained in fault domain $\mathcal{U}_1^A$, since all states can be reached via at most one transition from L' or U'. Clearly $\mathcal{S} \not\approx \mathcal{M}$, as input sequence *cpcp* provides a counterexample. Nevertheless, $\mathcal{M}$ passes test suite T. Thus the test suite generated by the SPYH-method [37] is not 1-A-complete.

Experiments. To explore whether the new fault domains $\mathcal{U}_k^A$ make sense from a practical perspective, we analyzed a large collection of FSM models of client and server implementations of the DTLS, TLS, SSH and EDHOC protocols. For clients and servers of these protocols there are usually a few sequences of interactions that occur during a regular run, and inputs that deviate from this "happy flow" may drive an implementation to some sink state. For each protocol P, we added the inputs from prefixes of the happy flow to a corresponding set A_P, together with an input sequence that typically leads to a sink state.

Janssen [20] inferred FSM models of more than 200 different versions of two major TLS server implementations. These models all share the same set of 11 inputs and their number of states varies from 6 to 14. We computed the eccentricity of the models relative to a set A_{TLS}, which happens to be a minimal state cover for all the models with 6 states. Appendix B presents the detailed results of our analysis. The eccentricity of the TLS models turns out to be at most 3, and therefore all the models are contained in the fault domain $\mathcal{U}_3^{A_{TLS}}$. This means that when a tester is asked to test some TLS server implementation, it is not unreasonable for him/her to hypothesize that this implementation is also contained in $\mathcal{U}_3^{A_{TLS}}$. As we will see, $\mathcal{U}_3^{A_{TLS}}$-complete test suites are way smaller than $\mathcal{U}_{14}$-complete test suites for a specification with 6 states.

4 A Sufficient Condition for *k*-*A*-Completeness

In this section, we describe a sufficient condition for a test suite to be k-A-complete, which (based on ideas of [6,40,41]) is phrased entirely in terms of properties of its testing tree. This tree should contain access sequences for each state in the specification, successors for these states for all possible inputs should be present up to depth $k + 1$, and apartness relations between certain states of the tree should hold. Before we present our condition and its correctness proof, we first need to introduce the concepts of apartness, basis and stratification.

In our condition, the concept of *apartness*, a constructive form of inequality, plays a central role [38,16].

Definition 4.1 (Apartness). *For a Mealy machine $\mathcal{M}$, we say that states $q, r \in Q^{\mathcal{M}}$ are* apart *(written $q \# r$) iff there is some $\sigma \in I^*$ such that $[\![q]\!](\sigma)\downarrow$, $[\![r]\!](\sigma)\downarrow$, and $[\![q]\!](\sigma) \neq [\![r]\!](\sigma)$. We say that σ is a* separating sequence *for q and r. We also say σ is a* witness *of $q \# r$ and write $\sigma \vdash q \# p$.*

Note that the apartness relation $\# \subseteq Q \times Q$ is irreflexive and symmetric. For the observation tree of Figure 2 we may derive the following apartness pairs and corresponding witnesses: $p \vdash 0 \# 1$ and $p \vdash 0 \# 11$. Observe that when two states are apart they are not equivalent, but states that are not equivalent are not necessarily apart. States 0 and 12, for instance, are neither equivalent nor apart. However, for complete Mealy machines apartness coincides with inequivalence.

In each observation tree, we may identify a basis: an ancestor closed set of states that are pairwise apart. In general, a basis is not uniquely determined, and an observation tree may have different bases with the same size. However, once we have fixed a basis, the remaining states in the tree can be uniquely partitioned by looking at their distance from the basis.

Definition 4.2 (Basis). *Let $\mathcal{T}$ be an observation tree. A nonempty subset of states $B \subseteq Q^{\mathcal{T}}$ is called a* basis *of $\mathcal{T}$ if*

1. *B is ancestor-closed: for all $q \in B : q \neq q_0^{\mathcal{T}} \;\Rightarrow\; \mathsf{parent}(q) \in B$, and*
2. *states in B are pairwise apart: for all $q, q' \in B : q \neq q' \;\Rightarrow\; q \# q'$.*

For each state q of $\mathcal{T}$, the candidate set *$C(q)$ is the set of basis states that are not apart from q: $C(q) = \{q' \in B \mid \neg(q \# q')\}$. State q is* identified *if $|C(q)| = 1$.*

Since B is nonempty and ancestor-closed, all states on the access path of a basis state are in the basis as well. In particular, the initial state $q_0^{\mathcal{T}}$ is in the basis. Also note that, by definition, basis states are identified.

A basis B induces a stratification of observation tree $\mathcal{T}$: first we have the set F^0 of immediate successors of basis states that are not basis states themselves, next the set F^1 of immediate successors of states in F^0, etc. In general, F^k contains all states that can be reached via a path of length $k+1$ from B.

Definition 4.3 (Stratification). *Let $\mathcal{T}$ be an observation tree with basis B. Then B induces a* stratification *of $Q^{\mathcal{T}}$ as follows. For $k \geq 0$,*

$$F^k = \{q \in Q^{\mathcal{T}} \mid d(B, q) = k + 1\}.$$

*We call F^k the k-*level frontier *and write $F^{<k} = \bigcup_{i<k} F^i$ and $F^{\leq k} = \bigcup_{i \leq k} F^i$.*

Example 4.4. Figure 4 shows the stratification for an observation tree for specification $\mathcal{S}$ from Figure 1 induced by basis $B = \{0, 1, 8\}$. Witness aa shows that the three basis states are pairwise apart, and therefore identified. States from sets B, F^0, F^1 and F^2 are marked with different colors. In Figure 4, B is complete, but F^0, F^1 and F^2 are incomplete (since states of F^0 and F^1 have no outgoing b-transitions, and states of F^2 have no outgoing transitions at all). The four F^0 states are also identified since $C(2) = \{0\}$, $C(5) = \{8\}$, $C(9) = \{0\}$, and $C(12) = \{1\}$. Two states in F^1 are identified since $C(3) = C(10) = \{1\}$, whereas the other two are not since $C(6) = C(13) = \{0, 8\}$. Since states in F^2 have no outgoing transitions, they are not apart from any other state, and thus $C(4) = C(7) = C(11) = C(14) = \{0, 1, 8\}$.

We are now prepared to state our characterization theorem.

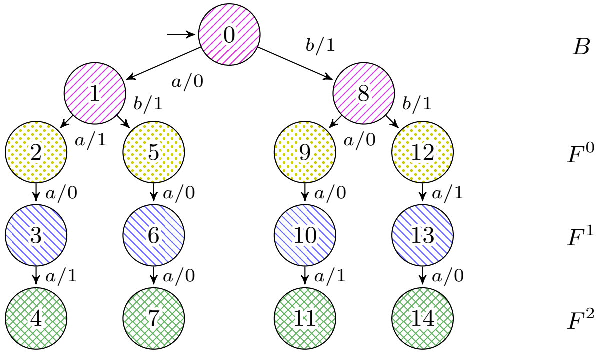

Fig. 4: Stratification of an observation tree induced by $B = \{0, 1, 8\}$.

Theorem 4.5. *Let $\mathcal{M}$ and $\mathcal{S}$ be Mealy machines, let $\mathcal{T}$ be an observation tree for both $\mathcal{M}$ and $\mathcal{S}$, let B be a basis for $\mathcal{T}$ with $|B| = |Q^{\mathcal{S}}|$, let $F^0, F^1, \ldots$ be the stratification induced by B, and let $k \geq 0$. Suppose B and $F^{<k}$ are complete, all states in F^k are identified, and the following condition holds:*

$$\forall q \in F^k \ \forall r \in F^{<k} : \ C(q) = C(r) \vee q \,\#\, r \tag{1}$$

Suppose that for every state q of $\mathcal{M}$ there are sequences $\sigma \in \mathsf{access}(B)$ and $\rho \in I^{\leq k}$ such that q is reached by $\sigma\rho$. Then $\mathcal{S} \approx \mathcal{M}$.

As a corollary, we obtain a sufficient condition for k-A-completeness.

Corollary 4.6. *Let $\mathcal{S}$ be a Mealy machine, let T be a test suite for $\mathcal{S}$, let $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$, let B be a basis for $\mathcal{T}$ with $|B| = |Q^{\mathcal{S}}|$, let $A = \mathsf{access}(B)$, let $F^0, F^1, \ldots$ be the stratification of $\mathcal{T}$ induced by B, and let $k \geq 0$. Suppose B and $F^{<k}$ are complete, all states in F^k are identified, and condition (1) holds. Then T is k-A-complete.*

The conditions of Corollary 4.6 do not only impose restrictions on testing tree $\mathcal{T}$, but also on specification $\mathcal{S}$. Suppose that the conditions of Corollary 4.6 hold. Then, by Lemma 2.9, there is a functional simulation $f : \mathcal{T} \to \mathcal{S}$. By Lemma A.8, f restricted to B is a bijection, $\mathcal{S}$ is minimal, and $\mathsf{access}(B)$ is a minimal state cover for $\mathcal{S}$. Furthermore, since B is complete and f restricted to B is a bijection, $\mathcal{S}$ is also complete. Minimality and completeness of specifications are common assumptions in conformance testing.

Example 4.7. A simple example of the application of Corollary 4.6, is provided by the observation tree from Figure 4 for the specification $\mathcal{S}$ from Figure 1. This observation tree corresponds to the test suite $T = \{aaaa, abaa, baaa, bbaa\}$. We claim that this test suite is 0-A-complete, for $A = \{\epsilon, a, b\} = \mathsf{access}(B)$. Note that condition (1) vacuously holds when $k = 0$. All other conditions of the theorem are also met: basis B is complete and all states in F^0 are identified. Therefore, according to Corollary 4.6, test suite T is 0-A-complete. We may slightly optimize the test suite by removing state 14 from the testing tree (i.e., replacing test *bbaa* by test *bba*), since all conditions of the theorem are still met for the reduced tree.

Note that Theorem 4.5 and Corollary 4.6 do not require that states in $F^{<k}$ are identified. As it turns out, the other conditions of the theorem/corollary already imply that states in $F^{<k}$ are identified.

Proposition 4.8. *Let $\mathcal{T}$ be an observation tree for $\mathcal{S}$, B a basis for $\mathcal{T}$ with $|B| = |Q^{\mathcal{S}}|$, $F^0, F^1, \ldots$ the stratification induced by B, and $k \geq 0$. Suppose B and $F^{<k}$ are complete, all states in F^k are identified, and condition (1) holds. Then all states in $F^{<k}$ are identified*

Appendix C shows that the converse implication does not hold.

As a consequence of the next proposition, condition (1) of Theorem 4.5 can be equivalently formulated as

$$\forall q \in F^k \; \forall r \in F^{<k} \; \forall s \in B : s \# q \;\Rightarrow\; s \# r \vee q \# r \tag{2}$$

Condition (2) says that apartness is *co-transitive* for triples of states in the observation tree consisting of a state in F^k, a state in $F^{<k}$, and a basis state. Co-transitivity is a fundamental property of apartness [38,16].

Proposition 4.9. *Let $\mathcal{T}$ be an observation tree for $\mathcal{S}$, B a basis for $\mathcal{T}$, and $|B| = |Q^{\mathcal{S}}|$. Suppose q and r are states of $\mathcal{T}$ and q is identified. Then*

$$C(q) = C(r) \vee q \# r \quad\Leftrightarrow\quad [\forall s \in B : s \# q \;\Rightarrow\; s \# r \vee q \# r]$$

Algorithm. We will now present Algorithm 1 that checks, for a given observation tree $\mathcal{T}$ with N states, in $\Theta(N^2)$ time, for all pairs of states, whether they are apart or not. Algorithm 1 assumes a total order $<$ on the set of inputs I and two partial functions $in : Q^{\mathcal{T}} \rightharpoonup I$ and $out : Q^{\mathcal{T}} \rightharpoonup O$ which, for each noninitial state q, specify the input and output, respectively, of the unique incoming transition of q. It also assumes, for each state $q \in Q^{\mathcal{T}}$, an adjacency list $Adj[q] \in (Q^{\mathcal{T}})^*$ that contains the immediate successors of q, sorted on their inputs. The algorithm maintains two Boolean arrays *Visited* and *Apart* to record whether a pair of states (q, q') has been visited or is apart, respectively. Initially, all entries in both arrays are *false*. When exploring a pair of states (q, q'), Algorithm 1 searches for outgoing transitions of q and q' with the same input label. In this case, if the outputs are different then it concludes that q and q' are apart. Otherwise, if the outputs are the same, it considers the target states r and r'. If the pair (r, r') has not been visited yet then the algorithm recursively explores whether this pair of states is apart. If r and r' are apart then q and q' are apart as well.

The theorem below asserts the correctness and time complexity of Algorithm 1.

Theorem 4.10. *Algorithm 1 terminates with running time $\Theta(N^2)$, where $N = |Q^{\mathcal{T}}|$. Upon termination, $Apart(q, q') = true$ iff $q \# q'$, for all $q, q' \in Q^{\mathcal{T}}$.*

Algorithm 1 Computing the apartness relation

```
function FillApartnessArray( )
    for $q \in Q$ do
        for $q' \in Q$ do
            if $\neg\, \mathit{Visited}(q, q')$ then ApartnessCheck$(q, q')$
            end if
        end for
    end for
    return $\mathit{Apart}$
end function
function ApartnessCheck$(q, q')$
    $l \leftarrow \mathit{Adj}[q]$, $l' \leftarrow \mathit{Adj}[q']$
    while $l \neq \epsilon \wedge l' \neq \epsilon \wedge \neg \mathit{Apart}(q, q')$
        $r \leftarrow \mathsf{hd}(\mathsf{l})$, $r' \leftarrow \mathsf{hd}(l')$
        if $\mathit{in}(r) < \mathit{in}(r')$ then $l \leftarrow \mathsf{tl}(l)$
        else if $\mathit{in}(r') < \mathit{in}(r)$ then $l' \leftarrow \mathsf{tl}(l')$
        else if $\mathit{out}(r) = \mathit{out}(r')$ then
            if $\neg\, \mathit{Visited}(r, r')$ then ApartnessCheck$(r, r')$
            end if
            $\mathit{Apart}(q, q') \leftarrow \mathit{Apart}(q, q') \vee \mathit{Apart}(r, r')$
            $l \leftarrow \mathsf{tl}(l)$, $l' \leftarrow \mathsf{tl}(l')$
        else$\mathit{Apart}(q, q') \leftarrow \mathit{true}$
        end if
    end while
    $\mathit{Visited}(q, q') \leftarrow \mathit{true}$
end function
```

Once we know the apartness relation, there are many things we can do, e.g., (1) we may check in $\Theta(N^2)$ time whether the conditions of Theorem 4.5 hold (but note that N grows exponentially in k), (2) similar to the H-method [6], we can select appropriate state identifiers on-the-fly in order to generate small k-A-complete test suites, (3) we can also check in $\Theta(N^2)$ time whether tests can be removed from a given test suite without compromising k-A-completeness.

5 Deriving Completeness for Existing Methods

In this section, we show how k-A-completeness of two popular algorithms for test suite generation, the Wp and HSI methods, follows from Corollary 4.6. We also present an alternative m-completeness proof of the H-method [6], which is a minor variation of the proof of Theorem 4.5. In order to define the Wp and HSI methods, we need certain sets (of sets) of sequences. The following definitions are based on [26]. We refer to [5,26] for a detailed exposition.

Definition 5.1. *Let $\mathcal{S}$ be a Mealy machine. A* state identifier *for a state $q \in Q^{\mathcal{S}}$ is a set $W_q \subseteq I^*$ such that for every inequivalent state $r \in Q^{\mathcal{S}}$, W_q contains a separating sequence for q and r. We write $\{W_q\}_{q \in Q^{\mathcal{S}}}$ or simply $\{W_q\}_q$ for a set* *that contains a state identifier W_q for each $q \in Q^{\mathcal{S}}$. If $\mathcal{W} = \{W_q\}_q$ is a set of state identifiers, then the* flattening $\bigcup \mathcal{W}$ *is the set* $\{\sigma \in I^* \mid \exists q \in Q^{\mathcal{S}} : \sigma \in W_q\}$*. If W is a set of input sequences and $\mathcal{W} = \{W_q\}_q$ is a set of state identifiers, then the* concatenation $W \odot \mathcal{W}$ *is defined as* $\{\sigma\tau \mid \sigma \in W,\ \tau \in W_{\delta^{\mathcal{S}}(q_0^{\mathcal{S}},\sigma)}\}$*. A set of state identifiers $\{W_q\}_q$ is* harmonized *if, for each pair of inequivalent states $q, r \in Q^{\mathcal{S}}$, $W_q \cap W_r$ contains a separating sequence for q and r. We refer to such a set of state identifiers as a* separating family*.*

The next proposition asserts that k-A-completeness of the Wp-method of Fujiwara et al [14] follows from Corollary 4.6 via routine checking.

Proposition 5.2 (k-A-completeness of the Wp-method). *Let $\mathcal{S}$ be a complete, minimal Mealy machine, $k \geq 0$, A a minimal state cover for $\mathcal{S}$, and $\mathcal{W} = \{W_q\}_q$ a set of state identifiers. Then $T = A \cdot I^{\leq k+1} \cup A \cdot I^{\leq k} \cdot \bigcup \mathcal{W} \cup A \cdot I^{\leq k+1} \odot \mathcal{W}$ is a k-A-complete test suite for $\mathcal{S}$.*

Also k-A-completeness of the HSI-method of Luo et al [24] and Petrenko et al [44,32] follows from Corollary 4.6 via a similar argument.

Proposition 5.3 (k-A-completeness of the HSI-method). *Let $\mathcal{S}$ be a complete, minimal Mealy machine, $k \geq 0$, A a minimal state cover for $\mathcal{S}$, and $\mathcal{W}$ a separating family. Then $T = A \cdot I^{\leq k+1} \cup A \cdot I^{\leq k+1} \odot \mathcal{W}$ is k-A-complete for $\mathcal{S}$.*

The W-method of [42,4], and the UIOv-method of [3] are instances of the Wp-method, and the ADS-method of [22] and the hybrid ADS method of [36] are instances of the HSI-method. This means that, indirectly, k-A-completeness of these methods also follows from our results.

The H-method of Dorofeeva et al [6] is based on a variant of our Theorem 4.5 which requires that all states in $F^{\leq k}$ are identified and replaces condition (1) by

$$\forall q, r \in F^{\leq k} : q \xrightarrow{+} r \Rightarrow C(q) = C(r) \vee q \# r \tag{3}$$

By Proposition A.11, condition (3) is implied by condition (1) of Theorem 4.5. Appendix D gives an example showing that the H-method is not k-A-complete. However, as shown by [6, Theorem 1], the H-method is m-complete. Our condition (1) can be viewed as a strengthening of condition (3) of the H-method, needed for k-A-completeness. Appendix E presents an alternative proof of m-completeness result for the H-method, using the proof technique of Theorem 4.5.

6 Conclusions and Future Work

We argued that the notion of m-complete test suites only has limited practical significance. As an alternative, we proposed the notion of k-A-completeness, which is based on a fault domain of FSMs in which all states can be reached by first performing a sequence from a state cover A for the specification, followed by k arbitrary inputs. We showed that the fault domain for k-A-completeness is larger than the one for m-completeness (if $m = |A| + k$), includes FSMs with a number of extra states that grows exponentially in k, and may be practically relevant in the setting of network protocols.

We provided a sufficient condition for k-A-completeness in terms of apartness of states of the observation/prefix tree induced by a test suite. Our condition can be checked efficiently (in terms of the size of the test suite) and can be used to prove k-A-completeness of the Wp-method of [14] and the HSI-method of [24,44,32]. Thus our results show that the Wp and HSI methods are complete for much larger fault domains than the ones for which they were originally designed. We presented counterexamples to show that the SPY-method [35], the H-method [6], and the SPYH-method [37] are not k-A-complete.

Hübner et al [19] investigated how test generation methods perform for SUTs whose behaviors lie outside the fault domain. They considered some realistic SUTs and used mutation operators to introduce bugs in a systematic and automated manner. Their experiments show that m-complete test suites generated by the W- and Wp-methods exhibit significantly greater test strength than conventional random testing, even for behavior outside the fault domain. It would be interesting to revisit these experiments and check which fraction of the detected faults is outside $\mathcal{U}_m$ but contained in $\mathcal{U}_k^A$.

Our condition is sufficient but not necessary for completeness. If one can prove, based on the assumptions of the selected fault domain, that two traces σ and τ reach the same state both in specification $\mathcal{S}$ and in implementation $\mathcal{M}$, then it makes no difference in test suites whether a suffix ρ is appended after σ or after τ. Simão et al [34,35] were the first to exploit this idea of *convergence* to reduce the size of test suites in a setting for m-completeness. It is future work to adapt these results to reduce the size of k-A-complete test suites. Also, the complexity of deciding whether a test suite is k-A-complete is still unknown.

Another direction for future work is to lift our results to partial FSMs with observable nondeterminism, e.g. by adapting the state counting method [32].

Closest to our characterization is the work of Sachtleben [33], who develops unifying frameworks for proving m-completeness of test generation methods. Inspired by the H-method, he defines an abstract H-condition that is sufficiently strong to prove m-completeness of the Wp, HSI, SPY, H and SPYH methods. Sachtleben [33] also considers partially defined FSMs with observable nondeterminism, and takes convergence into account. Moreover, he mechanized all his proofs using Isabelle. It would be interesting to explore whether the formalization of [33] can be adapted to our notion of k-A-completeness.

It will be interesting to explore if our characterization can be used to develop efficient test suite generation algorithms, or efficient algorithms for pruning test suites that have been generated by other methods. A promising research direction is to reduce the size of the fault domain via reasonable assumptions on the structure of the black box. Kruger et al [21] show that significantly smaller test suites suffice if the fault domains $\mathcal{U}_m$ are reduced by making plausible assumptions about the implementation. This approach can also be applied for the fault domains $\mathcal{U}_k^A$ proposed in this article.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
2. van den Bos, P., Janssen, R., Moerman, J.: n-Complete test suites for IOCO. Softw. Qual. J. **27**(2), 563–588 (2019). https://doi.org/10.1007/S11219-018-9422-X
3. Chan, W.Y.L., Vuong, C.T., Otp, M.R.: An improved protocol test generation procedure based on UIOs. In: Proceedings of the ACM Symposium on Communications Architectures & Protocols, September 19-22, 1989. p. 283–294. SIGCOMM '89, Association for Computing Machinery, New York, NY, USA (1989), https://doi.org/10.1145/75246.75274
4. Chow, T.: Testing software design modeled by finite-state machines. IEEE Trans. Software Eng. **4**(3), 178–187 (1978)
5. Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., Yevtushenko, N.: FSM-based conformance testing methods: A survey annotated with experimental evaluation. Information & Software Technology **52**(12), 1286–1297 (2010). https://doi.org/10.1016/j.infsof.2010.07.001
6. Dorofeeva, R., El-Fakih, K., Yevtushenko, N.: An improved conformance testing method. In: Wang, F. (ed.) Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3731, pp. 204–218. Springer (2005). https://doi.org/10.1007/11562436_16
7. Dragan, F.F., Leitert, A.: On the minimum eccentricity shortest path problem. Theoretical Computer Science **694**, 66–78 (2017). https://doi.org/https://doi.org/10.1016/j.tcs.2017.07.004
8. Fiterau-Brostean, P., Jonsson, B., Sagonas, K., Tåquist, F.: Automata-based automated detection of state machine bugs in protocol implementations. In: 30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023. The Internet Society (2023), https://www.ndss-symposium.org/ndss-paper/automata-based-automated-detection-of-state-machine-bugs-in-protocol-implementations/
9. Fiterău-Broştean, P., Jonsson, B., Sagonas, K., Tåquist, F.: Smbugfinder: An automated framework for testing protocol implementations for state machine bugs. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 1866–1870 (2024)
10. Fiterău-Broştean, P., Howar, F.: Learning-based testing the sliding window behavior of TCP implementations. *in* FMICS, LNCS **10471**, 185–200 (2017), https://doi.org/10.1007/978-3-319-67113-0_12
11. Fiterău-Broştean, P., Janssen, R., Vaandrager, F.: Combining model learning and model checking to analyze TCP implementations. In: Chaudhuri, S., Farzan, A. (eds.) Proceedings 28th International Conference on Computer Aided Verification (CAV'16), Toronto, Ontario, Canada. Lecture Notes in Computer Science, vol. 9780, pp. 454–471. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_25
12. Fiterău-Broştean, P., Jonsson, B., Merget, R., de Ruiter, J., Sagonas, K., Somorovsky, J.: Analysis of DTLS implementations using protocol state fuzzing. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2523–2540. USENIX Association (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/fiterau-brostean
13. Fiterău-Broştean, P., Lenaerts, T., Poll, E., de Ruiter, J., Vaandrager, F., Verleg, P.: Model learning and model checking of SSH implementations. In: Proceedings

of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. pp. 142–151. SPIN 2017, ACM, New York, NY, USA (2017), https://doi.org/10.1145/3092282.3092289

14. Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Transactions on Software Engineering **17**(6), 591–603 (1991). https://doi.org/10.1109/32.87284
15. Gazda, M., Hierons, R.M.: Model independent refusal trace testing. Science of Computer Programming **239**, 103173 (2025). https://doi.org/https://doi.org/10.1016/j.scico.2024.103173
16. Geuvers, H., Jacobs, B.: Relating apartness and bisimulation. Logical Methods in Computer Science **Volume 17, Issue 3** (Jul 2021). https://doi.org/10.46298/lmcs-17(3:15)2021
17. Hennie, F.C.: Fault detecting experiments for sequential circuits. In: Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design. pp. 95–110 (1964). https://doi.org/10.1109/SWCT.1964.8
18. Howar, F., Steffen, B.: Active automata learning in practice. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers. pp. 123–148. Springer International Publishing (2018)
19. Hübner, F., Huang, W., Peleska, J.: Experimental evaluation of a novel equivalence class partition testing strategy. Softw. Syst. Model. **18**(1), 423–443 (2019). https://doi.org/10.1007/S10270-017-0595-8
20. Janssen, E.: Fingerprinting TLS Implementations using Model Learning. Master thesis, Radboud University Nijmegen (Mar 2021), https://www.sidnlabs.nl/downloads/2eEQaXhsxKO0Js3FKoV2Yt/3136d1f6e7d60a1712e8e032631f7aca/Fingerprinting_TLS_Implementations_Using_Model_Learning_-_Erwin_Janssen.pdf
21. Kruger, L., Junges, S., Rot, J.: Small test suites for active automata learning. In: Finkbeiner, B., Kovács, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14571, pp. 109–129. Springer (2024). https://doi.org/10.1007/978-3-031-57249-4_6
22. Lee, D., Yannakakis, M.: Testing finite-state machines: State identification and verification. IEEE Trans. Comput. **43**(3), 306–320 (1994)
23. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines — a survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)
24. Luo, G., Petrenko, A., von Bochmann, G.: Selecting test sequences for partially-specified nondeterministic finite state machines. In: Mizuno, T., Higashino, T., Shiratori, N. (eds.) Protocol Test Systems: 7th workshop 7th IFIP WG 6.1 international workshop on protocol text systems. pp. 95–110. Springer US, Boston, MA (1995), https://doi.org/10.1007/978-0-387-34883-4_6
25. Maarse, T.: Active Mealy Machine Learning using Action Refinements. Master thesis, Radboud University Nijmegen (Aug 2020), https://www.cs.ru.nl/F.Vaandrager/thesis-2019_NWI-IMC048_s4416295.pdf
26. Moerman, J.: Nominal Techniques and Black Box Testing for Automata Learning. Ph.D. thesis, Radboud University Nijmegen (Jul 2019)

27. Moore, E.: Gedanken-experiments on sequential machines. In: Automata Studies. Annals of Mathematics Studies, vol. 34, pp. 129–153. Princeton University Press (1956)
28. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) 5^{th} GI Conference. Lecture Notes in Computer Science, vol. 104, pp. 167–183. Springer-Verlag (1981)
29. Peled, D., Vardi, M.Y., Yannakakis, M.: Black box checking. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Proceedings FORTE. IFIP Conference Proceedings, vol. 156, pp. 225–240. Kluwer (1999)
30. Petrenko, A., Higashino, T., Kaji, T.: Handling redundant and additional states in protocol testing. In: Cavalli, A., Budkowski, S. (eds.) Proceedings of the 8th International Workshop on Protocol Test Systems IWPTS '95, Paris, France. pp. 307–322 (1995)
31. Petrenko, A., Yevtushenko, N.: Conformance tests as checking experiments for partial nondeterministic FSM. In: Grieskamp, W., Weise, C. (eds.) Formal Approaches to Software Testing, 5th International Workshop, FATES 2005, Edinburgh, UK, July 11, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3997, pp. 118–133. Springer (2005). https://doi.org/10.1007/11759744_9
32. Petrenko, A., Yevtushenko, N., Lebedev, A., Das, A.: Nondeterministic state machines in protocol conformance testing. In: Rafiq, O. (ed.) Protocol Test Systems, VI, Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems, Pau, France, 28-30 September, 1993. IFIP Transactions, vol. C-19, pp. 363–378. North-Holland (1993)
33. Sachtleben, R.: Unifying frameworks for complete test strategies. Science of Computer Programming **237**, 103135 (2024). https://doi.org/10.1016/j.scico.2024.103135
34. da Silva Simão, A., Petrenko, A., Yevtushenko, N.: Generating reduced tests for FSMs with extra states. In: Núñez, M., Baker, P., Merayo, M.G. (eds.) Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5826, pp. 129–145. Springer (2009). https://doi.org/10.1007/978-3-642-05031-2_9
35. Simão, A., Petrenko, A., Yevtushenko, N.: On reducing test length for FSMs with extra states. Softw. Test. Verification Reliab. **22**(6), 435–454 (2012). https://doi.org/10.1002/STVR.452
36. Smeenk, W., Moerman, J., Vaandrager, F.W., Jansen, D.N.: Applying automata learning to embedded control software. In: Butler, M.J., Conchon, S., Zaïdi, F. (eds.) Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, France, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9407, pp. 67–83. Springer (2015). https://doi.org/10.1007/978-3-319-25423-4_5
37. Soucha, M., Bogdanov, K.: SPYH-method: An improvement in testing of finite-state machines. In: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops, Västerås, Sweden, April 9-13, 2018. pp. 194–203. IEEE Computer Society (2018). https://doi.org/10.1109/ICSTW.2018.00050
38. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2 edn. (2000). https://doi.org/10.1017/CBO9781139168717

39. Vaandrager, F.: Model learning. Communications of the ACM **60**(2), 86–95 (Feb 2017). https://doi.org/10.1145/2967606
40. Vaandrager, F.: A new perspective on conformance testing based on apartness. In: Capretta, V., Krebbers, R., Wiedijk, F. (eds.) Logics and Type Systems in Theory and Practice: Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday. pp. 225–240. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-61716-4_15
41. Vaandrager, F.W., Garhewal, B., Rot, J., Wißmann, T.: A new approach for active automata learning based on apartness. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13243, pp. 223–243. Springer (2022), https://doi.org/10.1007/978-3-030-99524-9_12
42. Vasilevskii, M.: Failure diagnosis of automata. Cybernetics and System Analysis **9**(4), 653–665 (1973). https://doi.org/https://doi.org/10.1007/BF01068590, (Translated from Kibernetika, No. 4, pp. 98-108, July-August, 1973.)
43. Yang, N., Aslam, K., Schiffelers, R.R.H., Lensink, L., Hendriks, D., Cleophas, L., Serebrenik, A.: Improving model inference in industry by combining active and passive learning. In: Wang, X., Lo, D., Shihab, E. (eds.) 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019. pp. 253–263. IEEE (2019), https://doi.org/10.1109/SANER.2019.8668007
44. Yevtushenko, N.V., Petrenko, A.F.: Synthesis of test experiments in some classes of automata. Autom. Control Comput. Sci. **24**(4), 50–55 (apr 1991)

A Proofs

We defined the lifted transition and output functions inductively for sequences of the form $i\sigma$. The next lemma basically says that we could have equivalently defined it inductively for sequences of the form σi.

Lemma A.1. *Let $\mathcal{M}$ be a Mealy machine, $q \in Q$, $i \in I$ and $\sigma \in I^*$. Then $\delta(q, \sigma i) = \delta(\delta(q, \sigma), i)$ and $\lambda(q, \sigma i) = \lambda(q, \sigma) \cdot \lambda(\delta(q, \sigma), i)$.*

In the proof of our completeness result, we use the concept of bisimulation relations and the well-known fact that for (deterministic) Mealy machines bisimulation equivalence coincides with the equivalence $\approx$ defined above.

Definition A.2 (Bisimulation). *A* bisimulation *between Mealy machines $\mathcal{M}$ and $\mathcal{N}$ is a relation $R \subseteq Q^{\mathcal{M}} \times Q^{\mathcal{N}}$ satisfying, for all $q \in Q^{\mathcal{M}}$, $r \in Q^{\mathcal{N}}$, and $i \in I$,*

1. $q_0^{\mathcal{M}} \; R \; q_0^{\mathcal{N}}$,
2. $q \; R \; r$ *implies* $\delta^{\mathcal{M}}(q, i)\downarrow \;\Leftrightarrow\; \delta^{\mathcal{N}}(r, i)\downarrow$,
3. $q \; R \; r \wedge \delta^{\mathcal{M}}(q, i)\downarrow$ *implies* $\delta^{\mathcal{M}}(q, i) \; R \; \delta^{\mathcal{N}}(r, i)$ *and* $\lambda^{\mathcal{M}}(q, i) = \lambda^{\mathcal{N}}(r, i)$.

We write $\mathcal{M} \simeq \mathcal{N}$ if there exists a bisimulation relation between $\mathcal{M}$ and $\mathcal{N}$.

The next lemma, a variation of the classical result of [28], is easy to prove.

Lemma A.3. *Let $\mathcal{M}, \mathcal{N}$ be Mealy machines. Then $\mathcal{M} \simeq \mathcal{N}$ iff $\mathcal{M} \approx \mathcal{N}$.*

The next lemma, which follows via a simple inductive argument from the definitions, states that a functional simulation preserves the transition and output functions on words.

Lemma A.4. *Suppose $f \colon \mathcal{M} \to \mathcal{N}$. Suppose $q, q' \in Q^{\mathcal{M}}$, $\sigma \in I^*$ and $\rho \in O^*$. Then $q \xrightarrow{\sigma/\rho} q'$ implies $f(q) \xrightarrow{\sigma/\rho} f(q')$.*

Proof. Suppose $q \xrightarrow{\sigma/\rho} q'$. Then $\lambda^{\mathcal{M}}(q, \sigma) = \rho$ and $\delta^{\mathcal{M}}(q, \sigma) = q'$. n order to establish $f(q) \xrightarrow{\sigma/\rho} f(q')$, $\lambda^{\mathcal{N}}(f(q), \sigma) = \rho$ and $\delta^{\mathcal{N}}(f(q), \sigma) = f(q')$. We prove both equalities by induction on the length of σ.

- Assume $\sigma = \epsilon$. Then
$$\lambda^{\mathcal{N}}(f(q), \sigma) = \lambda^{\mathcal{N}}(f(q), \epsilon) = \epsilon = \lambda^{\mathcal{M}}(q, \epsilon) = \lambda^{\mathcal{M}}(q, \sigma) = \rho,$$
$$\delta^{\mathcal{N}}(f(q), \sigma) = \delta^{\mathcal{N}}(f(q), \epsilon) = f(q) = f(\delta^{\mathcal{M}}(q, \epsilon)) = f(\delta^{\mathcal{M}}(q, \sigma)) = f(q').$$
- Assume $\sigma = i\sigma'$. We have $\delta^{\mathcal{M}}(q, i\sigma') = \delta^{\mathcal{M}}(\delta^{\mathcal{M}}(q, i), \sigma')$ and $\lambda^{\mathcal{M}}(q, i\sigma') = \lambda^{\mathcal{M}}(q, i)\lambda^{\mathcal{M}}(\delta^{\mathcal{M}}(q, i), \sigma')$. Let $q'' = \delta^{\mathcal{M}}(q, i)$, $o = \lambda^{\mathcal{M}}(q, i)$ and $\rho' = \delta^{\mathcal{M}}(q'', \sigma')$. Then we obtain $\rho = o\rho'$, $q \xrightarrow{i/o} q''$ and $q'' \xrightarrow{\sigma'/\rho'} q'$. Since f is a functional simulation and by the induction hypothesis, $f(q) \xrightarrow{i/o} f(q'')$ and $f(q'') \xrightarrow{\sigma'/\rho'} f(q')$. Then
$$\lambda^{\mathcal{N}}(f(q), \sigma) = \lambda^{\mathcal{N}}(f(q), i)\lambda^{\mathcal{N}}(\delta^{\mathcal{N}}(f(q), i), \sigma') = o\lambda^{\mathcal{N}}(f(q''), \sigma') = o\rho' = \rho$$
$$\delta^{\mathcal{N}}(f(q), \sigma) = \delta^{\mathcal{N}}(\delta^{\mathcal{N}}(f(q), i), \sigma') = \delta^{\mathcal{N}}(f(q''), \sigma') = f(q')$$
as required.

Proof of Lemma 2.9

We check the three conditions of Definition 2.4:

1. $f(q_0^{\mathcal{T}}) = f(\epsilon) = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \epsilon) = q_0^{\mathcal{S}}$.
2. Assume $\sigma \in Q^{\mathcal{T}}$ and $i \in I$ such that $\delta^{\mathcal{T}}(\sigma, i)\downarrow$. Then by Lemma A.1:
$$f(\delta^{\mathcal{T}}(\sigma, i)) = f(\sigma i) = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma i) = \delta^{\mathcal{S}}(\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma), i) = \delta^{\mathcal{S}}(f(\sigma), i).$$
3. Assume $\sigma \in Q^{\mathcal{T}}$ and $i \in I$ such that $\lambda^{\mathcal{T}}(\sigma, i)\downarrow$. Then
$$\lambda^{\mathcal{T}}(\sigma, i) = \lambda^{\mathcal{S}}(\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma), i) = \lambda^{\mathcal{S}}(f(\sigma), i).$$

Proof of Lemma 2.10

We prove both implications:

- Suppose $\mathcal{M}$ passes T. We prove $f \colon \mathcal{T} \to \mathcal{M}$ by checking the three conditions of Definition 2.4:
  1. $f(q_0^{\mathcal{T}}) = f(\epsilon) = \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \epsilon) = q_0^{\mathcal{M}}$.
  2. Assume $\delta^{\mathcal{T}}(\sigma, i)\downarrow$, for some $\sigma \in Q^{\mathcal{T}}$ and $i \in I$. Then by Lemma A.1:
  $$f(\delta^{\mathcal{T}}(\sigma, i)) = f(\sigma i) = \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma i) = \delta^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma), i) = \delta^{\mathcal{M}}(f(\sigma), i)$$
  3. As $\mathcal{M}$ passes T, for all $\sigma \in T$, $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) = \lambda^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$. This implies that, for all $\sigma i \in \mathit{Pref}(T)$, $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma i) = \lambda^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma i)$. By Lemma A.1, this implies
  $$\lambda^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma), i) = \lambda^{\mathcal{S}}(\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma), i).$$
  Now assume $\lambda^{\mathcal{T}}(\sigma, i)\downarrow$, for some $\sigma \in Q^{\mathcal{T}}$ and $i \in I$. Then $\sigma i \in \mathit{Pref}(T)$, and therefore, by the above equation,
  $$\lambda^{\mathcal{T}}(\sigma, i) = \lambda^{\mathcal{S}}(\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma), i) = \lambda^{\mathcal{M}}(\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma), i) = \lambda^{\mathcal{M}}(f(\sigma), i).$$
- Suppose $f \colon \mathcal{T} \to \mathcal{M}$. Let $\sigma \in T$. By construction, σ is also a state of $\mathcal{T}$ and, for some ρ, $\epsilon \xrightarrow{\sigma/\rho} \sigma$. By Lemma 2.9 and Lemma A.4, $q_0^{\mathcal{S}} \xrightarrow{\sigma/\rho} \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$. By the assumption and Lemma A.4, $q_0^{\mathcal{M}} \xrightarrow{\sigma/\rho} \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma)$. Hence $\lambda^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) = \rho = \lambda^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$.

Proof of Theorem 3.6

Suppose $\mathcal{M} \in \mathcal{U}_m \setminus \mathcal{U}^A$. We must show that $\mathcal{M} \in \mathcal{U}_k^A$. Since $\mathcal{M} \notin \mathcal{U}^A$, the states of $\mathcal{M}$ reached by sequences from A are pairwise inequivalent. Therefore, the set B of states of $\mathcal{M}$ that can be reached by sequences from A contains $|A|$ elements. Also, since $\epsilon \in A$ we know that $q_0^{\mathcal{M}} \in B$. Let q be a state of $\mathcal{M}$. Since we only consider Mealy machines that are initially connected, there exists a sequence of inputs σ that reaches q. Now pick σ in such a way that the number of states outside B visited by σ is minimal. Observe that each state outside B is visited at most once by σ (otherwise there would be a loop and the number of visited states outside B would not be minimal). Since $\mathcal{M}$ has at most m states and B contains $|A|$ states, this means that σ visits at most k states outside B. Since at least one state in B is visited (namely $q_0^{\mathcal{M}}$), this means that q can be reached with a sequence $\rho \in I^{\leq k}$ from a state in B. This implies that $\mathcal{M} \in \mathcal{U}_k^A$, as required.

The apartness of states $q \# r$ expresses that there is a conflict in their semantics, and consequently, apart states can never be identified by a functional simulation:

Lemma A.5. *For a functional simulation $f \colon \mathcal{T} \to \mathcal{M}$,*

$$q \# r \text{ in } \mathcal{T} \qquad \Longrightarrow \qquad f(q) \# f(r) \text{ in } \mathcal{M} \qquad \text{for all } q, r \in Q^{\mathcal{T}}.$$

Thus, whenever states are apart in the observation tree $\mathcal{T}$, the learner knows that these are distinct states in the hidden Mealy machine $\mathcal{M}$.

The apartness relation satisfies a weaker version of *co-transitivity*, stating that if $\sigma \vdash r \# r'$ and q has the transitions for σ, then q must be apart from at least one of r and r', or maybe even both:

Lemma A.6 (Weak co-transitivity). *In every Mealy machine $\mathcal{M}$,*

$$\sigma \vdash r \# r' \;\wedge\; \delta(q,\sigma){\downarrow} \quad\Longrightarrow\quad r \# q \;\vee\; r' \# q \qquad \text{for all } r, r', q \in Q^{\mathcal{M}}, \sigma \in I^*.$$

The next four lemmas give some useful properties of the basis.

Lemma A.7. *Suppose $\mathcal{T}$ is an observation tree for $\mathcal{M}$ with $f \colon \mathcal{T} \to \mathcal{M}$ and basis B. Then f restricted to B is injective.*

Proof. Let q and q' be two distinct states in B. Since $q \# q'$, we may conclude by Lemma A.5 that $f(q) \# f(q')$. Thus in particular $f(q) \neq f(q')$ and so f restricted to B is injective.

Lemma A.8. *Suppose $\mathcal{T}$ is an observation tree for $\mathcal{M}$ with $f \colon \mathcal{T} \to \mathcal{M}$ and basis B such that $|B| = |Q^{\mathcal{M}}|$. Then f restricted to B is a bijection, $\mathcal{M}$ is minimal, and $\mathsf{access}(B)$ is a minimal state cover for $\mathcal{M}$.*

Proof. By Lemma A.7, f restricted to B is injective. Since $|B| = |Q^{\mathcal{M}}|$, we may conclude that f is a bijection between B and $Q^{\mathcal{M}}$. Since states in B are pairwise apart, it follows by Lemma A.5 that states from $Q^{\mathcal{M}}$ are pairwise apart. This means that $\mathcal{M}$ is minimal. Since every state in B is reached by a unique sequence in $\mathsf{access}(B)$, and f is a bijection, we may use Lemma A.4 to conclude that also every state in $Q^{\mathcal{M}}$ is reached by a unique sequence in $\mathsf{access}(B)$.

Lemma A.9. *Suppose $\mathcal{T}$ is an observation tree for $\mathcal{M}$ with $f \colon \mathcal{T} \to \mathcal{M}$ and basis B such that $|B| = |Q^{\mathcal{M}}|$. Let q be a state of $\mathcal{T}$. Then there exists a state $r \in B$ with $r \in C(q)$ and $f(q) = f(r)$.*

Proof. Let $f(q) = u$. By Lemma A.8, f restricted to B is a bijection. Let $r \in B$ be the unique state with $f(r) = u$. Since $f(q) = f(r)$, Lemma A.5 implies that q and r are not apart. Hence $r \in C(q)$.

Lemma A.10. *Let $\mathcal{S}$ and $\mathcal{M}$ be Mealy machines, let T be a test suite for $\mathcal{S}$ such that $\mathcal{M}$ passes T, let B be a basis for $\mathsf{Tree}(\mathcal{S}, T)$, and let $A = \mathsf{access}(B)$. Then $\mathcal{M} \notin \mathcal{U}^A$.*

Proof. Let $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$. By Lemma 2.10, $\mathcal{T}$ is an observation tree for $\mathcal{M}$. Let $f : \mathcal{T} \to \mathcal{M}$. Suppose $\sigma, \rho \in A$ with $\sigma \neq \rho$. Since $A = \mathsf{access}(B)$, $q = \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) \in B$ and $p = \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \rho) \in B$. Since $\mathcal{T}$ is a tree, $p \neq q$ and thus, as B is a basis for $\mathcal{T}$, $q \# p$. By Lemma A.5, $f(q) \# f(p)$. By Lemma A.4, $f(q) = \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma)$ and $f(p) = \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \rho)$. From this we infer $\delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \sigma) \not\approx \delta^{\mathcal{M}}(q_0^{\mathcal{M}}, \rho)$. This implies $\mathcal{M} \notin \mathcal{U}^A$.

Proof of Theorem 4.5

Let $f : \mathcal{T} \to \mathcal{S}$ and $g : \mathcal{T} \to \mathcal{M}$. Define relation $R \subseteq Q^{\mathcal{S}} \times Q^{\mathcal{M}}$ by

$$(s, q) \in R \quad \Leftrightarrow \quad [\exists t \in B \cup F^{<k} : f(t) = s \wedge g(t) = q].$$

We claim that R is a bisimulation between $\mathcal{S}$ and $\mathcal{M}$, as defined in Definition A.2.

1. Since f is a functional simulation from $\mathcal{T}$ to $\mathcal{S}$, $f(q_0^{\mathcal{T}}) = q_0^{\mathcal{S}}$, and since g is a functional simulation from $\mathcal{T}$ to $\mathcal{M}$, $g(q_0^{\mathcal{T}}) = q_0^{\mathcal{M}}$. Using $q_0^{\mathcal{T}} \in B$, this implies $(q_0^{\mathcal{S}}, q_0^{\mathcal{M}}) \in R$.
2. Suppose $(s, q) \in R$ and $i \in I$. Since $(s, q) \in R$, there is a $t \in B \cup F^{<k}$ such that $f(t) = s$ and $g(t) = q$. Since B and $F^{<k}$ are complete, $\delta^{\mathcal{T}}(t, i)\downarrow$. Since f and g are functional simulations, also $\delta^{\mathcal{S}}(s, i)\downarrow$ and $\delta^{\mathcal{M}}(q, i)\downarrow$. Let $s' = \delta^{\mathcal{S}}(s, i)$, $q' = \delta^{\mathcal{M}}(q, i)$ and $t' = \delta^{\mathcal{T}}(t, i)$. Since f and g are functional simulations, $\lambda^{\mathcal{T}}(t, i) = \lambda^{\mathcal{S}}(s, i)$ and $\lambda^{\mathcal{T}}(t, i) = \lambda^{\mathcal{M}}(q, i)$. This implies $\lambda^{\mathcal{S}}(s, i) = \lambda^{\mathcal{M}}(q, i)$, as required. Since f and g are functional simulations, $f(t') = s'$ and $g(t') = q'$. In order to prove $(s', q') \in R$, we consider two cases:
   (a) $t' \in B \cup F^{<k}$. In this case, since $f(t') = s'$ and $g(t') = q'$, $(s', q') \in R$ follows from the definition of R.
   (b) $t' \in F^k$. In this case, by the final assumption, there are sequences $\sigma \in \mathsf{access}(B)$ and $\rho \in I^{\leq k}$ such that q' is reached by $\sigma\rho$. By the assumption that B and $F^{<k}$ are complete, $t'' = \delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma\rho)$ is defined. By Lemma A.4, $g(t'') = q'$. Then, by Lemma A.5, t' and t'' are not apart. We claim that t' and t'' have the same candidate set:
      i. $t'' \in B$. By definition of basis B, $C(t'') = \{t''\}$. Since not $t' \# t''$ and t' is identified, $C(t') = \{t''\}$. Hence $C(t'') = C(t')$.
      ii. $t'' \in F^{<k}$. Then by condition (1) and since not $t' \# t''$, $C(t'') = C(t')$.

   Since t' is identified, $C(t') = \{r\}$, for some $r \in B$. By Lemma A.9, $f(t') = f(r)$. Since $C(t'') = C(t')$, also $C(t'') = \{r\}$. Applying Lemma A.9 again gives $f(t'') = f(r)$. Hence $f(t'') = f(t') = s'$. This in turn implies $(s', q') \in R$, which completes the proof that R is bisimulation.

The theorem now follows by application of Lemma A.3.

Proof of Corollary 4.6

Let $\mathcal{M}$ be a Mealy machine in $\mathcal{U}_k^A \cup \mathcal{U}^A$. Assume that $\mathcal{M}$ passes T. In order to prove that T is k-A-complete, it suffices to prove that $\mathcal{M} \approx \mathcal{S}$. By Lemma 2.10, $\mathcal{T}$ is an observation tree for both $\mathcal{M}$ and $\mathcal{S}$. By Lemma A.10, $\mathcal{M} \not\in \mathcal{U}^A$. Therefore, $\mathcal{M} \in \mathcal{U}_k^A$. By definition of $\mathcal{U}_k^A$, for every state of $\mathcal{M}$ there are $\sigma \in A = \mathsf{access}(B)$ and $\rho \in I^{\leq k}$ such that q is reached by $\sigma\rho$. Now the result follows by Theorem 4.5.

Proof of Proposition 4.8

Proof by contradiction. Assume that some state $q \in F^{<k}$ is not identified. Then there are distinct states $r, s \in B$ such that $\{r, s\} \subseteq C(q)$. By definition of a basis, states r and s are apart. Let σ witness the apartness of r and s. Using that $F^{<k}$ is complete, we rerun σ from state q until we reach F^k. If $\delta^{\mathcal{T}}(q, \sigma) \in F^{\leq k}$, then by the weak co-transitivity Lemma A.6, q is apart from r or from s, and we have a contradiction. Otherwise, let ρ be the proper prefix of σ with $\delta^{\mathcal{T}}(q, \rho) \in F^k$. If $\rho \vdash r \# s$ then, by using Lemma A.6 again, we infer that q is either apart from r or from s, and we have a contradiction. So we may assume $\rho \not\vdash r \# s$. Let $\delta^{\mathcal{T}}(q, \rho) = q'$, $\delta^{\mathcal{T}}(r, \rho) = r'$ and $\delta^{\mathcal{T}}(s, \rho) = s'$. Since $\sigma \vdash r \# s$ but $\rho \not\vdash r \# s$, we conclude $r' \# s'$. Observe that both r' and s' are contained in $B \cup F^{<k}$. If both r' and s' are in B then, since q' is identified, q' is apart from r' or from s'. If $q' \# r'$ then $q \# r$ and we have a contradiction. If $q' \# s'$ then $q \# s$ and we have a contradiction. Otherwise, w.l.o.g., assume $r' \in F^{<k}$. Then, by condition (1), $C(q') = C(r')$ or $q' \# r'$. If $q' \# r'$ then $q \# r$ and we have a contradiction. So we conclude $C(q') = C(r')$. Since q' is identified $C(q') = \{t\}$, for some $t \in B$. If $s' \in B$ then, since $C(r') = \{t\}$, $s' \neq t$ because otherwise r' and s' are not apart. This implies $q' \# s'$, which implies $q \# s$, which is a contradiction. If $s' \in F^{<k}$ then, by condition (1), $C(q') = C(s')$ or $q' \# s'$. If $q' \# s'$ then $q \# s$ and we have a contradiction. So we conclude $C(q') = C(s') = \{t\}$. Let $f\colon \mathcal{T} \to \mathcal{S}$. By Lemma A.9, $f(r') = f(t) = f(s')$. But since $r' \# s'$, Lemma A.5 gives $f(r') \neq f(s')$. Contradiction.

Proof of Proposition 4.9

- "$\Rightarrow$" Assume $C(q) = C(r) \vee q \# r$. Suppose $s \in B$ with $s \# q$. We need to show $s \# r \vee q \# r$. By our assumption, if $a \# r$ then we are done. So suppose $C(q) = C(r)$. Then, since $s \# q$, $s \not\in C(q)$. Therefore $s \not\in C(r)$, which implies $s \# r$, as required.
- "$\Leftarrow$" Assume $\forall s \in B : s \# q \;\Rightarrow\; s \# r \vee q \# r$. Suppose not $q \# r$. We need to show $C(q) = C(r)$. Because q is identified, all basis states except one are apart from q. Let q' be the unique basis state that is not apart from q. By our assumption, r is apart from all states in $B \setminus \{q'\}$. Thus $C(r) \subseteq \{q'\}$. By Lemma A.9, $C(r)$ contains at least one state. Therefore, we conclude that $C(r) = \{q'\}$. This implies $C(q) = C(r)$, as required.

Condition (1) implies co-transitivity for a much larger collection of triples, namely triples of a basis state, a state in F^i and a state in F^j, for all $i \neq j$.

Proposition A.11. *Let $\mathcal{T}$ be an observation tree for $\mathcal{S}$, B a basis for $\mathcal{T}$ with $|B| = |Q^{\mathcal{S}}|$, $F^0, F^1, \ldots$ the stratification induced by B, and $k \geq 0$. Suppose B and $F^{<k}$ are complete, all states in F^k are identified, and condition (1) holds. Then $\forall i, j\ \forall q \in F^i\ \forall r \in F^j : 0 \leq i < j \leq k \quad \Rightarrow \quad C(q) = C(r) \vee q \# r$*

Proof. Note that, by the previous Proposition 4.8, all states in $F^{\leq k}$ are identified. Assume i and j are indices with $0 \leq i < j \leq k$, $q \in F^i$, $r \in F^j$ and $C(q) \neq C(r)$. It suffices to prove that $q \# r$. Let $f \colon \mathcal{T} \to \mathcal{S}$. Since both q and r are identified and $C(q) \neq C(r)$, it follows by Lemma A.9 and Lemma A.5 that $f(q) \# f(r)$. Let σ be a separating sequence for $f(q)$ and $f(r)$. If $\sigma \vdash q \# r$, we are done. Otherwise, since $F^{<k}$ is complete, there is a prefix ρ of σ with $\delta^{\mathcal{T}}(r, \rho) \in F^k$. Let $q' \in F^{<k}$ and $r' \in F^k$ be the unique states such that $q \xrightarrow{\rho} q'$ and $r \xrightarrow{\rho} r'$. If $\rho \vdash q \# r$, we are done. Otherwise, we conclude $f(q') \# f(r')$. By Lemma A.9 and since both q' and r' are identiied, we conclude $C(q') \neq C(r')$. Therefore, by condition (1), $q' \# r'$. Let τ be a witness for the apartness of q' and r'. Then $\rho\tau$ is a witness for the apartness of q and r. Thus $q \# r$, as required.

Proof of Theorem 4.10

Correctness follows since two states q and q' are apart if and only if either (1) both have an outgoing transition for the same input but with a different output, or (2) both have an outgoing transition for the same input, leading to states r and r', respectively, such that r and r' are apart. This is exactly what the algorithm checks.

Function ApartnessCheck is called exactly once for each pair of states (q, q'). The overall complexity of the algorithm is $\Theta(N^2)$, because $\mathcal{T}$ has $N-1$ transitions, and each pair of transitions (q, r) and (q', r') is considered at most once by the algorithm (during execution of ApartnessCheck(q, q')). The amount of work for each pair of transitions (q, r) and (q', r') is constant.

Proof of Proposition 5.2

Let $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$ and $f : \mathcal{T} \to \mathcal{S}$. Let B be the subset of states of $\mathcal{T}$ reached via an access sequence in A, and let $F^0, F^1, \ldots$ be the stratification of $Q^{\mathcal{T}}$ induced by B. We check that the assumptions of Corollary 4.6 hold:

1. Since $\mathcal{S}$ is complete, T is a test suite for $\mathcal{S}$.
2. B is a basis: Since A is a state cover for $\mathcal{S}$, it is prefix-closed. Hence, set B is ancestor-closed. Suppose q and q' are two distinct states in B. We show that $q \# q'$. Let σ and σ' be the access sequences of q and q', respectively. Then $\sigma \neq \sigma'$. Let $r = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$ and $r' = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma')$. By Lemma A.4, $f(q) = r$ and $f(q') = r'$. Since A is a minimal state cover, $r \neq r'$, and since $\mathcal{S}$ is minimal, $r \not\approx r'$. Set $\bigcup \mathcal{W}$ contains a separating sequence τ for r and r'. Since $A \cdot \bigcup \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(q, \rho) \downarrow$ and $\delta^{\mathcal{T}}(q', \rho) \downarrow$. By Lemma A.4, $\lambda^{\mathcal{T}}(q, \rho) = \lambda^{\mathcal{S}}(r, \rho) \neq \lambda^{\mathcal{S}}(r', \rho) = \lambda^{\mathcal{T}}(q', \rho)$. Thus $\rho \vdash q \# q'$, as required.
3. Since $A \subseteq T$, $|A| = |B|$, and since A is a minimal state cover, $|A| = |Q^{\mathcal{S}}|$. Hence $|B| = |Q^{\mathcal{S}}|$.

4. By construction, $A = \mathsf{access}(B)$.
5. Since $A \cdot I^{\leq k+1} \subseteq T$, sets B and $F^{<k}$ are complete.
6. All states in F^k are identified: We show a stronger statement, namely that all states in $F^{\leq k}$ are identified. Suppose $r \in F^{\leq k}$. Let $\mathsf{access}(r) = \sigma$ and $s = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$. By Lemma A.4, $f(r) = s$. By Lemma A.8, f restricted to B is a bijection. Let $q \in B$ be the unique state with $f(q) = s$. Now suppose $q' \in B$ is distinct from q. Let $f(q') = s'$. By definition of a state identifier, W_s contains a separating sequence ρ for s and s'. Since $A \cdot I^{\leq k+1} \odot \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(r, \rho) \downarrow$. Since $A \cdot \bigcup \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(q', \rho) \downarrow$. By Lemma A.4,
$$\lambda^{\mathcal{T}}(q', \rho) = \lambda^{\mathcal{S}}(s', \rho) \neq \lambda^{\mathcal{S}}(s, \rho) = \lambda^{\mathcal{T}}(r, \rho).$$
Thus $\rho \vdash r \# q'$. Since q' was chosen to be an arbitrary basis state different from q, this implies that r is identified.
7. Condition (1) holds: Suppose $q \in F^k$ and $r \in F^{<k}$. By the previous item, both q and r are identified, that is, there exist $q', r' \in B$ such that $C(q) = \{q'\}$ and $C(r) = \{r'\}$. If $q' = r'$ then $C(q) = C(r)$ and we are done. So assume $q' \neq r'$. Let $s = f(q')$ and $t = f(r')$. By Lemma A.9, $f(q) = s$ and $f(r) = t$. Since f restricted to B is a bijection, $s \neq t$, and since $\mathcal{S}$ is minimal, $s \not\approx t$. By definition of a state identifier, W_s contains a separating sequence ρ for s and t. Since $A \cdot I^{\leq k+1} \odot \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(q, \rho) \downarrow$. Since $A \cdot I^{\leq k} \cdot \bigcup \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(r, \rho) \downarrow$. By Lemma A.4, $\lambda^{\mathcal{T}}(q, \rho) = \lambda^{\mathcal{S}}(s, \rho) \neq \lambda^{\mathcal{S}}(t, \rho) = \lambda^{\mathcal{T}}(r, \rho)$. Thus $\rho \vdash r \# q$.

Since all conditions of Corollary 4.6 hold, we conclude that T is k-A-complete.

Proof of Proposition 5.3

Let $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$ and $f : \mathcal{T} \to \mathcal{S}$. Let B be the subset of states of $\mathcal{T}$ reached via an access sequence in A, and let $F^0, F^1, \ldots$ be the stratification of $Q^{\mathcal{T}}$ induced by B. We check that the assumptions of Corollary 4.6 hold:

1. Since $\mathcal{S}$ is complete, T is a test suite for $\mathcal{S}$.
2. B is a basis: Since A is a state cover for $\mathcal{S}$, it is prefix-closed. Hence set B is ancestor-closed. Suppose q and q' are two distinct states in B. We show that $q \# q'$. Let σ and σ' be the access sequences of q and q', respectively. Then $\sigma \neq \sigma'$. Let $r = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$ and $r' = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma')$. By Lemma A.4, $f(q) = r$ and $f(q') = r'$. Since A is a minimal state cover, $r \neq r'$, and since $\mathcal{S}$ is minimal, $r \not\approx r'$. Since $\mathcal{W} = \{W_q\}_q$ is a separating family, $W_r \cap W_{r'}$ contains a separating sequence ρ for r and r'. Since $A \odot \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(q, \rho) \downarrow$ and $\delta^{\mathcal{T}}(q', \rho) \downarrow$. By Lemma A.4, $\lambda^{\mathcal{T}}(q, \rho) = \lambda^{\mathcal{S}}(r, \rho) \neq \lambda^{\mathcal{S}}(r', \rho) = \lambda^{\mathcal{T}}(q', \rho)$. Thus $\rho \vdash q \# q'$, as required.
3. Since $A \subseteq T$, $|A| = |B|$, and since A is a minimal state cover, $|A| = |Q^{\mathcal{S}}|$. Hence $|B| = |Q^{\mathcal{S}}|$.
4. By construction, $A = \mathsf{access}(B)$.
5. Since $A \cdot I^{\leq k+1} \subseteq T$, sets B and $F^{<k}$ are complete.

6. All states in F^k are identified: We show a stronger statement, namely that all states in $F^{\leq k}$ are identified. Suppose $r \in F^{\leq k}$. Let $\mathsf{access}(r) = \sigma$ and let $s = \delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma)$. By Lemma A.4, $f(r) = s$. By Lemma A.8, f restricted to B is a bijection. Let $q \in B$ be the unique state with $f(q) = s$. Now suppose $q' \in B$ is distinct from q, and let $f(q') = s'$. Since f is a bijection, $s \neq s'$, and since $\mathcal{S}$ is minimal $s \not\approx s'$. By the definition of a separating family, $W_s \cap W_{s'}$ contains a separating sequence ρ for s and s'. Since $A \cdot I^{\leq k+1} \odot \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(r, \rho)\downarrow$ and $\delta^{\mathcal{T}}(q', \rho)\downarrow$. By Lemma A.4, $\lambda^{\mathcal{T}}(q', \rho) = \lambda^{\mathcal{S}}(s', \rho) \neq \lambda^{\mathcal{S}}(s, \rho) = \lambda^{\mathcal{T}}(r, \rho)$. Thus $\rho \vdash r \# q'$. Since q' was chosen to be an arbitrary basis state different from q, this implies that state r is identified.
7. Condition (1) holds: Suppose $q \in F^k$ and $r \in F^{<k}$. By the previous item, both q and r are identified, that is, there exist $q', r' \in B$ such that $C(q) = \{q'\}$ and $C(r) = \{r'\}$. If $q' = r'$ then $C(q) = C(r)$ and we are done. So assume $q' \neq r'$. Let $s = f(q')$ and $t = f(r')$. By Lemma A.9, $f(q) = s$ and $f(r) = t$. Since f restricted to B is a bijection, $s \neq t$, and since $\mathcal{S}$ is minimal, $s \not\approx s'$. By the definition of a separating family, $W_s \cap W_t$ contains a separating sequence ρ for s and t. Since $A \cdot I^{\leq k+1} \odot \mathcal{W} \subseteq T$, $\delta^{\mathcal{T}}(q, \rho)\downarrow$ and $\delta^{\mathcal{T}}(r, \rho)\downarrow$. By Lemma A.4, $\lambda^{\mathcal{T}}(q, \rho) = \lambda^{\mathcal{S}}(s, \rho) \neq \lambda^{\mathcal{S}}(t, \rho) = \lambda^{\mathcal{T}}(r, \rho)$. Thus $\rho \vdash r \# q$.

Since all conditions of Corollary 4.6 hold, we conclude that T is k-A-complete.

B Experiments

To explore the practicality of the new fault domains, we computed eccentricity for a large number of network protocol implementation models which were generated by model learning. The set of access sequences A was from a prefix-closure of a set comprising a few common sequences of interaction traversing the main states in the respective protocol specification. We performed two experiments, one considering numerous implementations of four network protocols, and another considering many versions of two implementations of TLS (version 1.2). In our result tables, for each SUT Mealy machine $\mathcal{M}$ considered, we include the following measurements: the size of $\mathcal{M}$'s alphabet in terms of inputs ($|I|$); the size of $\mathcal{M}$ in terms of states ($|Q|$); the number of states in the basis ($|B|$), which is the set of distinct states in Q that sequences in A cover; the eccentricity result (ϵ), which is obtained by computing $\max_{q \in Q} d(B, q)$. We additionally specify the SUT for which $\mathcal{M}$ was generated, by indicating the library and where available, the version.

B.1 Measuring eccentricity for DTLS, TLS, SSH and EDHOC

In our first experiment, we considered 92 implementation models for DTLS (version 1.2), TLS (version 1.2), SSH, and EDHOC. For DTLS, SSH and EDHOC, we used the models included in artifacts provided by prior work [8,9]. For TLS, we obtained the models from the AutomataWiki (https://automata.cs.ru.nl/). We considered both server and (where available) client models. Overal, we found that with a k of six we could produce a fault domain $\mathcal{U}_k^A$ that includes all protocol implementation models. Most models could be captured with smaller values for k, as Table 1 shows. We continue by discussing the experimental setup and results for each protocol

Table 1: Number of protocol implementation models included in $\mathcal{U}_k^A$ for increasing values of k

k	0	1	2	3	4	5	6
Num. Models in $\mathcal{U}_k^A$	18	45	67	84	86	91	93

For DTLS, we used client and server models with alphabets using combinations of the DHE, ECDHE, PSK and RSA key exchange algorithms. Access sequences were derived from sequences completing handshakes using these algorithms. Table 2 shows the results, using additional columns to indicate the role (client/server) and the key exchange configuration the alphabet used. We see that with a relatively small value for k and some basis, we can often generate fault domains that capture models with many more states than those in the basis. For example, a k of three and a basis of 21 states, generated a fault domain that included the 43 state model of a GnuTLS 3.5.19 server. Similarly, the 24 state model of a WolfSSL 4.0.0 server is included in a fault domain generated using a k of two and a basis of 15 states.

For TLS, we used client and server models with two kinds of alphabets: one (we call **basic**) which uses the RSA key exchange algorithm, the other (we call **full**) which also uses DHE and additionally includes Heartbeat messages. Similarly to DTLS, access sequences in A were derived from sequences completing handshakes using the key exchange algorithms supported by the alphabets. To these, we added a sequence which lead to a disconnected state. Results are shown in Table 3. We see that a k of three is sufficient is generate fault domains that included all models, yet many models can be captured using a smaller k. For example, the server model of GnuTLS 3.3.8 using full alphabet has 16 states, yet it is included in a fault domain generated with k of only 1, and a basis with only 9 states. The corresponding 16 state client model is included in a fault model generated using similar parameters.

Our SSH experiment involved server models whose alphabets include inputs which are necessary to perform key exchange, authenticate, open and close a channel. Access sequences were derived from a happy flow sequence which performs the aforementioned steps, sequences which perform a second key exchange in states where this is permitted by the specification, and a sequence which leads to a disconnected state. Table 4 displays the results. All the models could be captured in fault models produced using a k of five. Remarkably, a k of three can generate a fault model which includes the 66 state model of BitVise 7.23.

For EDHOC, the alphabets included input necessary to complete a key exchange and send data. Access sequences were derived from two sequences

Table 2: Eccentricity calculations for DTLS. Alphabets are described by the combination of key exchange algorithms they use, which are abbreviated by their starting letter: D (DHE), E (ECDHE), P (PSK) and R (RSA).

Role	SUT	Alphabet Desc.	$\|I\|$	$\|Q\|$	$\|B\|$	ϵ
client	GnuTLS 3.6.7	D+E+R	20	22	15	2
client	GnuTLS 3.6.7	P	8	12	7	2
client	MbedTLS 2.16.1	D+E+R	20	18	17	1
client	MbedTLS 2.16.1	P	8	8	7	1
client	OpenSSL 1.1.1b	D+E+R	20	23	17	2
client	OpenSSL 1.1.1b	P	8	13	7	2
client	OpenSSL 1.1.1k	D+E+R	20	23	17	2
client	PionDTLS 1.5.2	E	16	49	14	6
client	PionDTLS 1.5.2	P	8	21	9	4
client	PionDTLS 2.0.2	E	16	5	4	1
client	PionDTLS 2.0.2	P	8	18	10	2
client	PionDTLS 2.0.9	P	8	18	10	2
client	PionDTLS usenix	E	16	49	14	6
client	PionDTLS usenix	P	8	21	9	4
client	Scandium 2.0.0-M16	P	8	11	6	2
client	Scandium 2.3.0	E	17	12	10	1
client	Scandium 2.6.2	E	17	12	10	1
client	Scandium 2.6.2	P	8	8	7	1
client	TinyDTLS^C	E	17	22	14	2
client	TinyDTLS^C	P	8	14	10	1
client	TinyDTLS^E	E	17	17	11	1
client	TinyDTLS^E	P	8	13	9	1
client	WolfSSL 4.0.0	P	8	13	10	1
client	WolfSSL 4.4.0	P	8	15	11	2
client	WolfSSL 4.7.1r	P	8	17	10	2
server	GnuTLS 3.5.19	D+E+P+R	16	43	21	3
server	GnuTLS 3.6.7	D+E+P+R	16	16	14	1
server	GnuTLS 3.7.1	D+E+P+R	16	16	14	1
server	MbedTLS 2.16.1	D+E+P+R	16	17	14	3
server	MbedTLS 2.26.0	D+E+P+R	16	17	14	3
server	OpenSSL 1.1.1b	D+E+P+R	16	19	15	2
server	OpenSSL 1.1.1k	D+E+P+R	16	23	15	3
server	PionDTLS 1.5.2	E	10	66	11	5
server	PionDTLS 1.5.2	P	7	14	8	1
server	PionDTLS 2.0.2	E	10	25	12	3
server	PionDTLS 2.0.2	P	7	16	9	2
server	PionDTLS 2.0.9	E	10	25	12	3
server	PionDTLS 2.0.9	P	7	16	9	2
server	PionDTLS usenix	E	10	66	11	5
server	PionDTLS usenix	P	7	14	8	1
server	Scandium 2.0.0-M16	E	10	38	10	5
server	Scandium 2.0.0-M16	P	7	16	8	3
server	Scandium 2.3.0	E	10	15	10	2
server	Scandium 2.3.0	P	7	13	8	2
server	Scandium 2.6.2	E	10	11	10	1
server	Scandium 2.6.2	P	7	9	8	1
server	TinyDTLS^C	E	10	30	14	3
server	TinyDTLS^C	P	7	25	10	3
server	TinyDTLS^E	E	10	27	14	3
server	TinyDTLS^E	P	7	22	10	3
server	WolfSSL 4.0.0	D+E+R	14	24	15	2

Table 3: Eccentricity calculations for TLS

Role	SUT	Alphabet Desc.	$\lvert I\rvert$	$\lvert Q\rvert$	$\lvert B\rvert$	ϵ
client	GnuTLS 3.3.12	full	12	9	9	0
client	GnuTLS 3.3.8	full	12	15	9	1
client	GnuTLS 3.3.8	regular	8	11	6	1
client	NSS 3.17.4	full	12	11	11	0
client	NSS 3.17.4	regular	8	7	7	0
client	OpenSSL 1.0.1g	regular	7	10	6	3
client	OpenSSL 1.0.1j	regular	7	6	6	0
client	OpenSSL 1.0.1l	regular	7	6	6	0
client	OpenSSL 1.0.2	full	10	9	9	0
client	OpenSSL 1.0.2	regular	7	6	6	0
server	GnuTLS 3.3.12	full	12	9	9	0
server	GnuTLS 3.3.12	regular	8	7	7	0
server	GnuTLS 3.3.8	full	11	16	9	1
server	GnuTLS 3.3.8	regular	8	12	7	1
server	NSS 3.17.4	regular	8	8	7	1
server	OpenSSL 1.0.1j	regular	7	11	6	3
server	OpenSSL 1.0.1l	regular	7	10	6	3
server	OpenSSL 1.0.2	regular	7	7	6	1
server	RSA BSAFE C 4.0.4	regular	8	9	6	1
server	RSA BSAFE Java 6.1.1	regular	8	6	6	0
server	miTLS 0.1.3	regular	8	6	6	0

Table 4: Eccentricity calculations for SSH servers

SUT	$\lvert I\rvert$	$\lvert Q\rvert$	$\lvert B\rvert$	ϵ
BitVise 7.23	13	66	25	3
BitVise 8.49	12	43	19	2
Dropbear v2014.65	13	17	13	1
Dropbear v2020.81	12	21	12	3
OpenSSH 8.2p1	12	37	15	5
OpenSSH 8.8p1	12	37	15	5

completing key exchange and one sequence causing the SUT to disconnect. Table 5 displays the results. A k of two was sufficient to generate fault models that include all 12 models. Seven of them are included in a k of zero.

Table 5: Eccentricity calculations for EDHOC

Role	SUT	$\|I\|$	$\|Q\|$	$\|B\|$	ϵ
client	lakers	9	3	3	0
client	RISE	9	3	3	0
client	RISE_m4_app	9	5	4	1
client	sifis-home_phase_1	9	5	3	2
client	uoscore-uedhoc_linux_edhoc	9	2	2	0
client	uoscore-uedhoc_linux_edhoc_oscore	9	4	4	0
server	lakers	9	3	3	0
server	RISE	9	3	2	1
server	RISE_m4_app	9	5	3	2
server	sifis-home_phase_1	9	5	3	2
server	uoscore-uedhoc_linux_edhoc	9	3	3	0
server	uoscore-uedhoc_linux_edhoc_oscore	9	4	4	0

B.2 Measuring eccentricity for many versions of two TLS libraries

Our second experiment considered 169 server models that were generated by Janssen [20] for various versions two widely used TLS libraries in OpenSSL and MbedTLS. The models use an alphabet with 11 inputs. We derived access sequences from a single sequence which completed the TLS handshake and then closed the connection. Results are shown in Table 6. The basis has six states for all models, hence we ommit it from the table. States in all models could be reached within at most three transitions. This was the case even for the 13-14 state models of the earlier versions of OpenSSL.

C Condition (1) is Needed

Proposition 4.8 establishes that condition (1) implies that all states in $F^{<k}$ are identified. The converse implication does not hold: even if all states in $B \cup F^{\leq k}$ are identified, condition (1) may not hold. The Mealy machines $\mathcal{S}$ and $\mathcal{M}$ of Figure 5 present a counterexample with $k = 1$ and $A = \{\epsilon, l, r\}$. Note that these machines are not equivalent: input sequence $rrrlll$ distinguishes them. The extra state q_3 of $\mathcal{M}$ behaves similar as state q_0 of $\mathcal{S}$, but is not equivalent. Figure 6 shows an observation tree $\mathcal{T}$ for both $\mathcal{S}$ and $\mathcal{M}$. Observation tree $\mathcal{T}$ meets all the requirements of Theorem 4.5, except condition (1). One way to think of $\mathcal{T}$ is that $\mathcal{M}$ cherry picks distinguishing sequences from $\mathcal{S}$ to ensure that the F^1 states are identified by a sequence for which $\mathcal{S}$ and $\mathcal{M}$ agree. Note that B and F^0 are both

Table 6: Eccentricity calculations for many versions of MbedTLS and OpenSSL

MbedTLS Versions	$\lvert Q\rvert$	ϵ
1.2.1 1.2.10 1.2.11 1.2.2 1.2.3 1.2.4 1.2.5 1.2.6 1.2.7 1.2.8 1.2.9 1.3.0 1.3.1 1.3.2 1.3.3 1.3.4 1.3.5 1.3.6 1.3.7 1.3.8 2.0.0 2.1.0 2.1.1 2.1.10 2.1.11 2.1.12 2.1.13 2.1.14 2.1.15 2.1.16 2.1.17 2.1.18 2.1.2 2.1.3 2.1.4 2.1.5 2.1.6 2.1.7 2.1.8 2.1.9 2.10.0	6	0
2.11.0 2.12.0 2.13.0 2.13.1 2.14.0 2.14.1 2.15.0 2.15.1 2.16.0 2.16.1 2.16.2 2.16.3 2.16.4 2.16.5 2.16.6 2.16.7 2.16.8 2.17.0 2.18.0 2.18.1 2.19.0 2.19.0d1 2.19.0d2 2.19.1	8	2
2.2.0 2.2.1	6	0
2.20.0 2.20.0d0 2.20.0d1 2.21.0 2.22.0 2.22.0d0 2.23.0 2.24.0	8	2
2.3.0 2.4.0 2.4.1 2.4.2 2.5.0 2.5.1 2.6.0 2.6.1 2.7.0 2.7.1 2.7.10 2.7.11 2.7.12 2.7.13 2.7.14 2.7.15 2.7.16 2.7.17 2.7.2 2.7.3 2.7.4 2.7.5 2.7.6 2.7.7 2.7.8 2.7.9 2.8.0 2.9.0	6	0
3.0.0p1	8	2

OpenSSL Versions	$\lvert Q\rvert$	ϵ
1.0.1 1.0.1a 1.0.1b 1.0.1c 1.0.1d	13	3
1.0.1d 1.0.1e 1.0.1f 1.0.1g	14	3
1.0.1h	13	3
1.0.1i 1.0.1j 1.0.1k 1.0.1l 1.0.1m 1.0.1n 1.0.1o 1.0.1p 1.0.1q 1.0.1r 1.0.1s 1.0.1t 1.0.1u	11	2
1.0.2 1.0.2a 1.0.2b 1.0.2c 1.0.2d 1.0.2e 1.0.2f 1.0.2g 1.0.2h 1.0.2i 1.0.2j 1.0.2k 1.0.2l	10	2
1.0.2m 1.0.2n 1.0.2o 1.0.2p 1.0.2q 1.0.2r 1.0.2s 1.0.2t 1.0.2u 1.1.0 1.1.0a 1.1.0b 1.1.0c 1.1.0d 1.1.0e 1.1.0f 1.1.0g 1.1.0h 1.1.0i 1.1.0j 1.1.0k 1.1.0l 1.1.1 1.1.1a 1.1.1b 1.1.1c 1.1.1d 1.1.1e 1.1.1f 1.1.1g	8	2

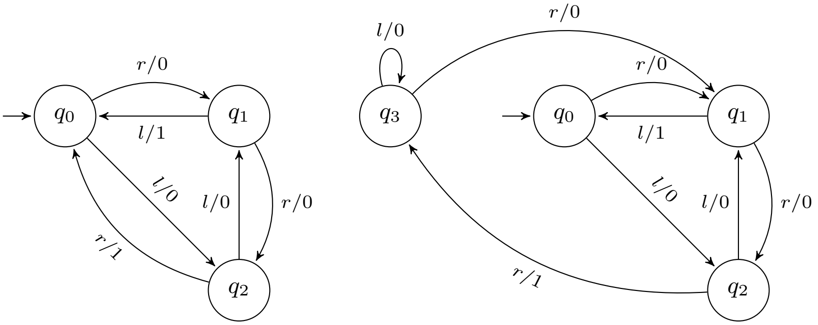

Fig. 5: A specification $\mathcal{S}$ (left) and a faulty implementation $\mathcal{M}$ (right).

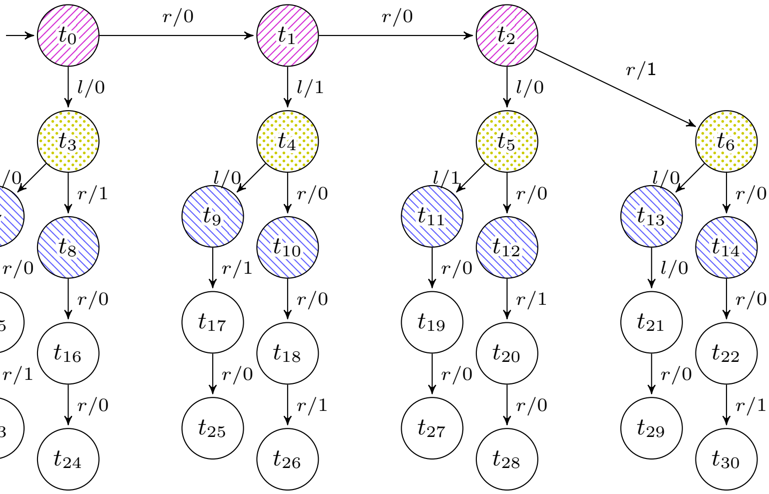

Fig. 6: Observation tree for FSMs $\mathcal{S}$ and $\mathcal{M}$ from Figure 5.

complete, and all states in B and $F^{\leq 1}$ are identified. However, the tree does not satisfy condition (1) as t_{13} is not apart from t_6, $C(t_6) = \{t_0\}$ and $C(t_{13}) = \{t_2\}$. The example shows that without condition (1), Theorem 4.5 does not hold.

D The SPY and H-methods are not k-A-Complete

Example D.1. Figure 7(left) shows the running example $\mathcal{S}$ from the article by Simão, Petrenko and Yevtushenko that introduces the SPY-method [35]. Using this method, a 3-complete test suite $\{aaaa, baababba, bbabaa\}$ was derived in [35]. Consider the minimal state cover $A = \{\epsilon, a\}$ for $\mathcal{S}$. Implementation $\mathcal{M}$ from Figure 7(right) is contained in fault domain $\mathcal{U}_1^A$, since all states can be reached via at most one transition from $0'$ and $1'$. Clearly $\mathcal{S} \not\approx \mathcal{M}$, as input sequence *aab* provides a counterexample. Nevertheless, $\mathcal{M}$ passes the derived test suite. Thus the test suite generated by the SPY-method [35] is not 1-A-complete.

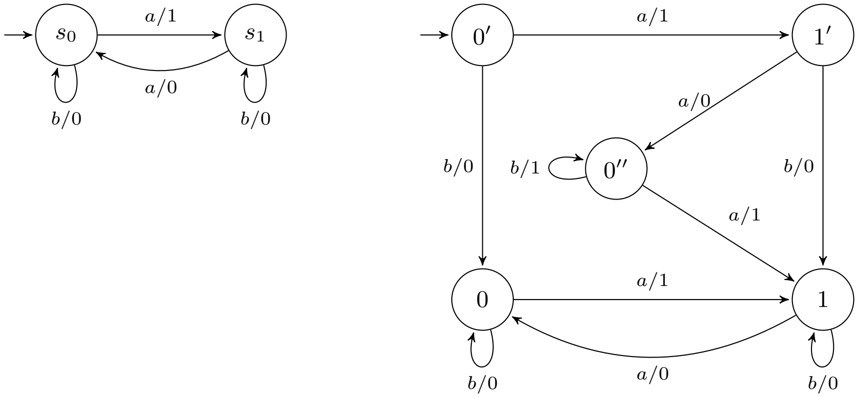

Fig. 7: An implementation from fault domain $\mathcal{U}_1^A$ (right) that passes the 3-complete test suite $\{aaaa, baababba, bbabaa\}$ that was constructed for the specification (left) using the SPY-method.

Example D.2. Figure 9 shows the tree for a 3-complete test suite generated by the H-method of Dorofeeva et al [6] for the machine of Figure 8(left). This tree satisfies condition (3) since the only transitions from F^0 to F^1 that change the candidate set are a-transitions, and the sources and targets of those transitions are apart. The machine of Figure 8(right) will pass this test suite, even though the two machines are inequivalent (*cbc* is a counterexample). It is easy to check that the machine on the right is in $\mathcal{U}_1^A$, for $A = \{\epsilon, a\}$. Thus the test suite generated by the H-method is not 1-A-complete. Indeed, the test suite does not meet condition (1) since (for example) the states with access sequences *cb* and *ac* have different candidate sets but are not apart.

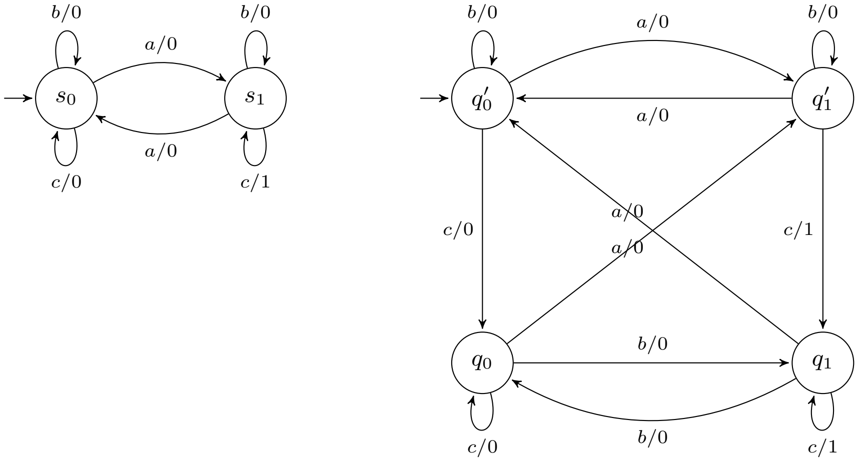

Fig. 8: Specification (left) and implementation (right) from fault domain $\mathcal{U}_1^A$ that passes the test suite of Figure 9.

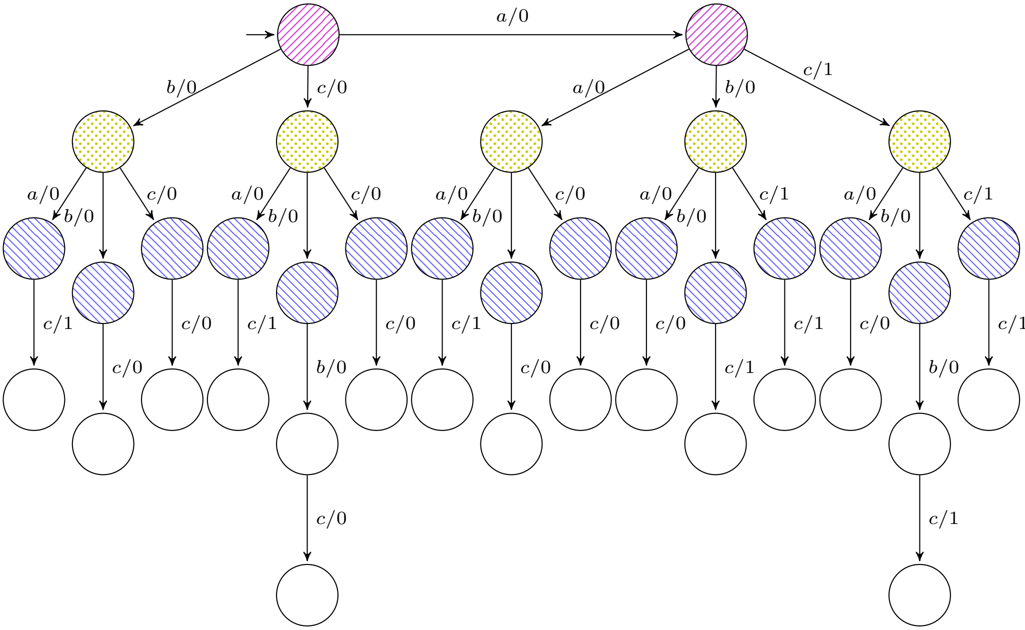

Fig. 9: Testing tree for 3-complete test suite constructed for the specification of Figure 8(left) using the H-method.

E The H-method is m-Complete

Below we present an alternative proof of m-completeness result for the H-method of [6], restated for our setting, using the same proof technique as Theorem 4.5.

Proposition E.1 (m-completeness of the H-method). *Let $\mathcal{S}$ be a Mealy machine with n states, let T be a test suite for $\mathcal{S}$, let $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$, let B be a basis for $\mathcal{T}$ with n states, let $A = \mathsf{access}(B)$, let $F^0, F^1, \ldots$ be the stratification of $\mathcal{T}$ induced by B, and let $m, k \geq 0$ with $m = n + k$. Suppose B and $F^{<k}$ are complete, all states in $F^{\leq k}$ are identified, and condition (3) holds:*

$$\forall q, r \in F^{\leq k} : q \xrightarrow{+} r \Rightarrow C(q) = C(r) \vee q \# r$$

Then T is m-complete.

Proof. Let $\mathcal{M}$ be a Mealy machine with at most m states that passes suite T. In order to prove that T is m-complete, it suffices to prove that $\mathcal{M} \approx \mathcal{S}$. Let $f : \mathcal{T} \to \mathcal{S}$ and $g : \mathcal{T} \to \mathcal{M}$. Define relation $R \subseteq Q^{\mathcal{S}} \times Q^{\mathcal{M}}$ by

$$(s, q) \in R \Leftrightarrow \exists t \in B \cup F^{<k} : f(t) = s \wedge g(t) = q.$$

We claim that R is a bisimulation between $\mathcal{S}$ and $\mathcal{M}$.

1. Since f is a functional simulation from $\mathcal{T}$ to $\mathcal{S}$, $f(q_0^{\mathcal{T}}) = q_0^{\mathcal{S}}$, and since g is a functional simulation from $\mathcal{T}$ to $\mathcal{M}$, $g(q_0^{\mathcal{T}}) = q_0^{\mathcal{M}}$. Using $q_0^{\mathcal{T}} \in B$, this implies $(q_0^{\mathcal{S}}, q_0^{\mathcal{M}}) \in R$.
2. Suppose $(s, q) \in R$ and $i \in I$. Since $(s, q) \in R$, there exists a $t \in B \cup F^{<k}$ such that $f(t) = s$ and $g(t) = q$. W.l.o.g. we select t from the lowest possible stratum, that is, if $t \in F^i$ then there is no $\bar{t} \in B \cup F^{<i}$ with $f(\bar{t}) = s$ and $g(\bar{t}) = q$. Since B and $F^{<k}$ are complete, $\delta^{\mathcal{T}}(t, i) \downarrow$. Since f and g are functional simulations, also $\delta^{\mathcal{S}}(s, i) \downarrow$ and $\delta^{\mathcal{M}}(q, i) \downarrow$. Let $s' = \delta^{\mathcal{S}}(s, i)$, $q' = \delta^{\mathcal{M}}(q, i)$ and $t' = \delta^{\mathcal{T}}(t, i)$. Since f and g are functional simulations, $\lambda^{\mathcal{T}}(t, i) = \lambda^{\mathcal{S}}(s, i)$ and $\lambda^{\mathcal{T}}(t, i) = \lambda^{\mathcal{M}}(q, i)$. This implies $\lambda^{\mathcal{S}}(s, i) = \lambda^{\mathcal{M}}(q, i)$, as required. Since f and g are functional simulations, $f(t') = s'$ and $g(t') = q'$. In order to prove $(s', q') \in R$, we consider two cases:
   (a) $t' \in B \cup F^{<k}$. In this case, since $f(t') = s'$ and $g(t') = q'$, $(s', q') \in R$ follows from the definition of R.
   (b) $t' \in F^k$. Let W be the subset of states in $F^{<k}$ that occur in the access path of t'. We claim that g is injective on $B \cup W$. Then, since $B \cup W$ has $n + k = m$ states, there must be a state $t'' \in B \cup W$ with $g(t'') = g(t')$. States t'' and t' have the same candidate set (otherwise they would be apart by condition (3) or by the fact that t' has been identified), which would contradict $g(t'') = g(t')$). Then by Lemma A.9, $f(t'') = f(t')$. This in turn implies that $(s', q') \in R$, which completes the proof that R is bisimulation.

   Thus it remains to prove our claim that g is injective on $B \cup W$:
   i. By Lemma A.7, g is injective on B.

ii. Let $u \in W$ and let $r \in B$. We claim $g(u) \neq g(r)$. We consider two cases:
   - $C(u) = \{r\}$. Then by Lemma A.9, $f(u) = f(r)$. But then $g(u) \neq g(r)$, because otherwise t would not be in the lowest possible stratum.
   - $C(u) \neq \{r\}$. Then $u \# r$ and therefore, by Lemma A.5, $g(u) \neq g(r)$.

iii. Let $u, u' \in W$ with $u \neq u'$. We claim $g(u) \neq g(u')$. By condition (3), $C(u) = C(u')$ or $u \# u'$.
   - If $C(u) = C(u')$, Lemma A.9 gives that $f(u) = f(u')$. But then $g(u) \neq g(u')$, because otherwise t would not be in the lowest possible stratum.
   - If $u \# u'$ then $g(u) \neq g(u')$ by Lemma A.5.

The proposition now follows by application of Lemma A.3.