New Fault Domains for Conformance Testing of Finite State Machines

Frits Vaandrager
Radboud University, Nijmegen, The Netherlands

Ivo Melse
Radboud University, Nijmegen, The Netherlands

Abstract

A fault domain reflects a 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 in the literature on black-box conformance testing is the class of finite state machines (FSMs) with at most m states. Numerous strategies for generating test suites have been proposed that guarantee fault coverage for this class. These so-called m-complete test suites grow exponentially in $m - n$, where n is the number of states of the specification, so one can only run them for small values of $m - n$. But the assumption that $m - n$ is small is not realistic in practice. In his seminal paper from 1964, Hennie raised the challenge to design checking experiments in which the number of states may increase appreciably. In order to solve this long-standing open problem, we propose (much larger) fault domains that capture the assumption that all states in an implementation 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. The number of states of FSMs in these fault domains grows exponentially in k. We present a sufficient condition for k-A-*completeness* of test suites with respect to these fault domains. Our condition implies k-A-completeness of two prominent m-complete test suite generation strategies, the Wp and HSI methods. Thus these strategies are complete for much larger fault domains than those for which they were originally designed, and thereby solve Hennie's challenge. We show that three other prominent m-complete methods (H, SPY and SPYH) do not always generate k-A-complete test suites.

2012 ACM Subject Classification Theory of computation

Keywords and phrases conformance testing, finite state machines, Mealy machines, apartness, observation tree, fault domains, k-A-complete test suites

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2025.34

Related Version *Full Version*: `https://arxiv.org/abs/2410.19405`

Funding Supported by NWO project OCENW.M.23.155, Evidence-Driven Black-Box Checking (EVI).

Acknowledgements This article grew out of earlier work of the first author [35, 37] and the bachelor thesis of the second author [21]. An early version was made available on arXiv [36]. Many thanks to Kirill Bogdanov, Paul Fiterău-Broştean, Dennis Hendriks, Bálint Kocsis, Bram Pellen, Jurriaan Rot, and the anonymous reviewers for their insightfull feedback on earlier versions.

1 Introduction

We revisit the classic problem of black-box conformance testing [18] 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 $\mathcal{S}$. 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. The challenge then becomes to define a fault domain that captures *realistic assumptions* about possible faults, but still allows for the design of *sufficiently small* test suites that are complete for the fault domain and can be run within reasonable time.

A fault domain that has been widely studied is the set $\mathcal{U}_m$ 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 [23] and Hennie [12]. Numerous methods for constructing m-complete test suites have been proposed, for different types of transition system models, see for instance [39, 3, 41, 9, 28, 26, 18, 6, 27, 30, 31, 32, 38, 22, 16, 35, 10]. We refer to [18, 5, 22] 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 extra states. This is problematic in practice, since the size of m-complete test suites generated by existing methods grows exponentially in $m - n$, where n is the number of states of the specification. Actually, Moore [23] was not aiming for practical methods and only described *gedanken-experiments.* He introduced the example of combination lock machines, and was therefore well aware of the combinatorial explosions in m-complete test suites. In his seminal paper from 1964, Hennie [12] 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 [39] and Chow [3] independently showed that m-complete test suites can be constructed with a size that is polynomial in the size of the specification FSM, for a fixed value of $k = m - n$. Nevertheless, the test suites generated by their W-method grow exponentially in k, and so they did not solve the problem raised by Hennie [12].

In this article, we solve the long-standing open problem of Hennie [12] as follows:

1. As an alternative for m-completeness, 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.
2. Based on ideas from [6, 35, 37], 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 [9], and the HSI-method of Luo et al [19] and Petrenko et al [41, 28]. The W-method of Vasilevskii [39] and Chow [3], and the UIOv-method of Chan et al [2] are instances of the Wp-method, and the ADS-method of Lee & Yannakakis [17] and the hybrid ADS method of Smeenk et al [31] are instances of the HSI-method. This means that 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 those for which they were designed originally.
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 [30] and the SPYH-method of Soucha and Bogdanov [32], do not always generate k-A-complete test suites.

The rest of this article is structured as follows. First, Section 2 recalls some basic definitions regarding (partial) Mealy machines, observation trees, and test suites. 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 (and its proof), Section 5 establishes k-A-completeness of the Wp and HSI methods, and m-completeness of the H-method. Finally, Section 6, discusses implications of our results and directions for future research. All proofs are deferred to the full version of this article available on arXiv.

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 sequence 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

Next, we recall the definition of Finite State Machines (FSMs) a.k.a. 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 $q \xrightarrow{i/o} q'$ 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*

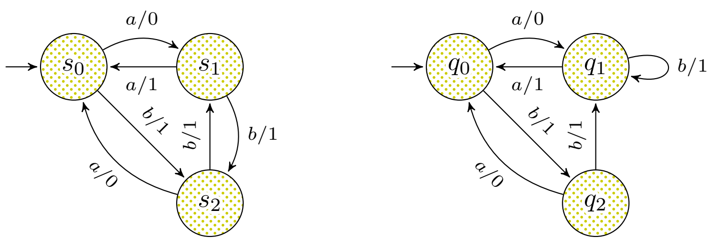

Figure 1 A specification $\mathcal{S}$ (left) and an inequivalent implementation $\mathcal{M}$ (right).

$\lambda(q,\sigma)=\rho$ 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. $\mathcal{M}$ 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 [22]) 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.

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, for $q \in Q^{\mathcal{M}}$ and $i \in I$,*

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

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 transition $q \xrightarrow{i/o} q'$ of $\mathcal{M}$ can be matched by a transition $f(q) \xrightarrow{i/o} f(q')$ 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 may 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 $\mathcal{M}$.*

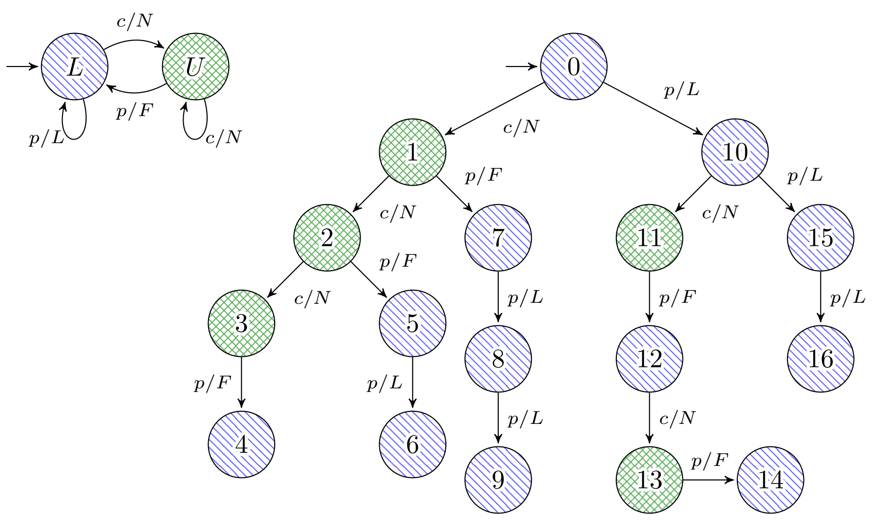

Figure 2 A Mealy machine $\mathcal{S}$ (left) and an observation tree $\mathcal{T}$ for $\mathcal{S}$ (right).

▶ **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 [32], 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 indicate the functional simulation from $\mathcal{T}$ to $\mathcal{S}$.

2.4 Test Suites

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

▶ **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, it also passes all 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:*

- *$Q^{\mathcal{T}} = \{\epsilon\} \cup \mathit{Pref}(T)$ 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

A *fault domain* 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)**.** *A* fault domain *is a set $\mathcal{U}$ of Mealy machines. A test suite T for a Mealy machine $\mathcal{S}$ is $\mathcal{U}$-*complete *if, for each $\mathcal{M} \in \mathcal{U}$, $\mathcal{M}$ passes T implies $\mathcal{M} \approx \mathcal{S}$.*

The next three lemmas are immediate consequences of the above definition. In particular, whenever T is $\mathcal{U}$-complete, completeness is preserved when we add tests to T or remove machines from $\mathcal{U}$.

▶ **Lemma 3.2.** *If T and T' are test suites with $T \subseteq T'$ and T is $\mathcal{U}$-complete, then T' is $\mathcal{U}$-complete.*

▶ **Lemma 3.3.** *If $\mathcal{U}$ and $\mathcal{U}'$ are fault domains with $\mathcal{U}' \subseteq \mathcal{U}$, and T is a $\mathcal{U}$-complete test suite, then T is $\mathcal{U}'$-complete.*

▶ **Lemma 3.4.** *If a test suite for $\mathcal{S}$ is both $\mathcal{U}$-complete and $\mathcal{U}'$-complete, then it is $\mathcal{U} \cup \mathcal{U}'$-complete.*

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

▶ **Definition 3.5.** *Let $m > 0$. Then fault domain $\mathcal{U}_m$ 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.

We consider fault domains of Mealy machines in which all states can be reached by first performing an access sequence from a set A (typically a state cover of specification $\mathcal{S}$), followed by 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 scenarios from A. The next definition formally introduces the corresponding fault domains $\mathcal{U}_k^A$. A somewhat similar notion was previously proposed by Maarse [20], in a specific context with action refinement.

▶ **Definition 3.6.** *Let k be a natural number and let $A \subseteq I^*$. Then fault domain $\mathcal{U}_k^A$ 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 3.7.** 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 two states L' and U' that are reachable via A.

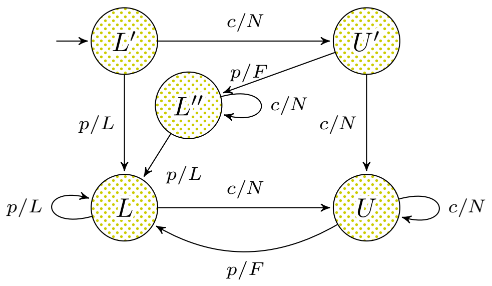

Figure 3 A Mealy machine $\mathcal{M}$ contained in fault domain $\mathcal{U}_1^A$, with $A = \{\epsilon, c\}$.

▶ Remark 3.8. 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 the maximum distance from w to any other vertex:

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

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. The *eccentricity* $\epsilon(W)$ of a set of vertices W is the maximum distance from W to any other vertex:

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

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.

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 leads us to the following proposition.

▶ **Proposition 3.9.** *Let $A \subset I^*$ be prefix closed with $|A| = n$, let $|I| = l$ and $k > 0$. Then fault domain $\mathcal{U}_k^A$ contains Mealy machines with up to $n + (\sum_{j=0}^{k-1} l^j)(nl - n + 1)$ states.*

This observation implies that the number of states of Mealy machines in $\mathcal{U}_k^A$ 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 [8]. 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 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 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. So these methods exclude any model $\mathcal{M}$ in which distinct sequences from A reach equivalent states. This motivates the following definition:

▶ **Definition 3.10.** *Let $A \subseteq I^*$. Then fault domain $\mathcal{U}^A$ 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, fault domain $\mathcal{U}_m$ is contained in fault domain $\mathcal{U}_k^A \cup \mathcal{U}^A$.

▶ **Theorem 3.11.** *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 $\mathcal{U}_m \subseteq \mathcal{U}_k^A \cup \mathcal{U}^A$.*

We refer to $\mathcal{U}_k^A \cup \mathcal{U}^A$-complete test suites as *k-A-complete*. By Theorem 3.11 and Lemma 3.3, any k-A-complete test suite is also m-complete, if $m = |A| + k$. The converse of the inclusion of Theorem 3.11 does not hold, as $\mathcal{U}^A$ may contain FSMs with an unbounded number of states. The next example illustrates that a test suite T can be m-complete, but not 0-A-complete for a state cover A of the specification.

▶ **Example 3.12.** The set $A = \{\epsilon, a, aa\}$ is a minimal state cover of Mealy machine $\mathcal{S}$ at the left of Figure 4. Machine $\mathcal{S}$ is minimal and aaa is a *distinguishing sequence* since it generates different outputs for each of the three states of $\mathcal{S}$. Mealy machine $\mathcal{S}$ is trivially contained in fault domains $\mathcal{U}_3$ and $\mathcal{U}_0^A$, whereas Mealy machine $\mathcal{M}$ at the right of Figure 4 is clearly not contained in $\mathcal{U}_3$ and $\mathcal{U}_0^A$.

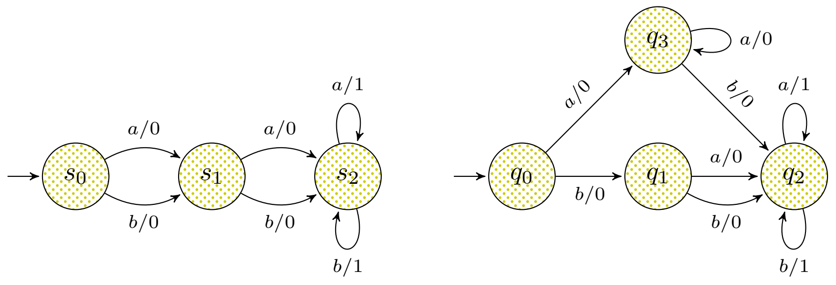

Figure 4 A specification $\mathcal{S}$ (left) and an inequivalent implementation $\mathcal{M}$ (right).

However, $\mathcal{M}$ is contained in fault domain $\mathcal{U}^A$ since $\delta^{\mathcal{M}}(q_0, a) = \delta^{\mathcal{M}}(q_0, aa)$. Note that the set $B = \{\epsilon, b, bb\}$ is also a minimal state cover for $\mathcal{S}$, and the sequence bbb is also a distinguishing sequence. Using the Wp-method, to be discussed in more detail in Section 5, 0-B-complete test suites can easily be built from a state cover and a distinguishing sequence. In particular, the set $T = B \cdot \{bbb\} \;\cup\; B \cdot \{a, b\} \cdot \{bbb\}$ is a 0-B-complete test suite for $\mathcal{S}$, and therefore (by Theorem 3.11) also 3-complete. However, since $\mathcal{M}$ passes test suite T, T is not $\mathcal{U}_0^A \cup \mathcal{U}^A$-complete, and thus not 0-A-complete.

Below we give an example to show that, for $k > 0$, m-complete test suites generated by the SPYH-method [32] are not always k-A-complete, if $m = |A| + k$. Appendix B contains variations of this example, which demonstrate that the SPY-method [30] and the H-method [6] are not k-A-complete either.

▶ **Example 3.13.** Consider specification $\mathcal{S}$ and testing tree $\mathcal{T}$ from Figure 2. This specification and the corresponding test suite $T = \{cccp, ccpp, cppp, pcpcp, ppp\}$ were both taken from [32], 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 $\mathcal{S}$. FSM $\mathcal{M}$ from Figure 3 belongs to 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 [32] is not 1-A-complete.

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, 35, 37]) 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, and study their basic properties.

4.1 Apartness

In our sufficient condition, the concept of *apartness*, inspired by a similar notion that is standard in constructive real analysis [33, 11], plays a central role.

▶ **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 call σ 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.

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 4.2.** *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 $\mathcal{T}$, we know that the corresponding states in Mealy machine $\mathcal{M}$ are distinct. 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 4.3** (Weak co-transitivity). *In every Mealy machine $\mathcal{M}$,*

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

4.2 Basis

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 for instance 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 this basis.

▶ **Definition 4.4** (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. The next lemma gives some useful properties of a basis.

▶ **Lemma 4.5.** *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 $\mathcal{M}$.*

Whenever a subset of states B of a testing tree is a basis, the corresponding test suite is $\mathcal{U}^A$-complete, for $A = \mathsf{access}(B)$.

▶ **Lemma 4.6.** *Let $\mathcal{S}$ be a Mealy machine, let T be a test suite for $\mathcal{S}$, let B be a basis for $\mathcal{T} = \mathsf{Tree}(\mathcal{S}, T)$, and let $A = \mathsf{access}(B)$. Then T is $\mathcal{U}^A$-complete.*

4.3 Stratification

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

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

*We call F^k the k-*level frontier *and write $F^{<k} = \bigcup_{0 \leq i < k} F^i$ and $F^{\leq k} = \bigcup_{0 \leq i \leq k} F^i$.*

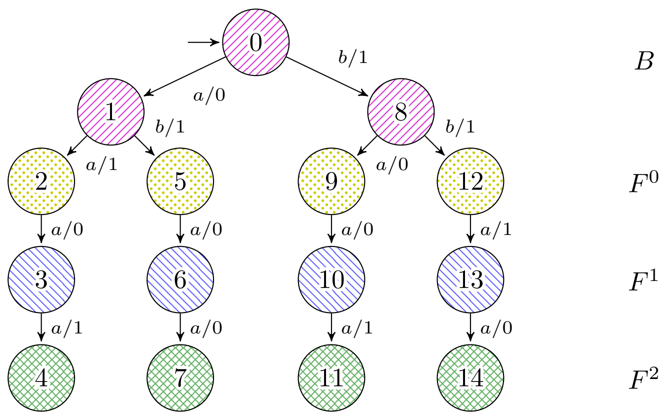

Figure 5 Stratification of an observation tree induced by $B = \{0, 1, 8\}$.

▶ **Example 4.8.** Figure 5 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 5, 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 $C(4) = C(7) = C(11) = C(14) = \{0, 1, 8\}$.

4.4 A Sufficient Condition for k-A-completeness

We are now prepared to state our characterization theorem.

▶ **Theorem 4.9.** *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 $A = \mathsf{access}(B)$, 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:*

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

Suppose that $\mathcal{M} \in \mathcal{U}_k^A$. Then $\mathcal{S} \approx \mathcal{M}$.

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

▶ **Corollary 4.10.** *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 4.10 do not only impose restrictions on testing tree $\mathcal{T}$, but also on specification $\mathcal{S}$. Suppose that the conditions of Corollary 4.10 hold. Then, by Lemma 2.9, there is a functional simulation $f : \mathcal{T} \to \mathcal{S}$. By Lemma 4.5, 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 4.11.** A simple example of the application of Corollary 4.10, is provided by the observation tree from Figure 5 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 Corollary 4.10 are also met: basis B is complete and all states in F^0 are identified. Therefore, test suite T is 0-A-complete. We may slightly optimize the test suite by replacing test $bbaa$ by test bba, since all conditions of the corollary are still met for the reduced testing tree.

The conditions of Corollary 4.10 are sufficient for k-A-completeness, but the following trivial example illustrates that they are not necessary.

▶ **Example 4.12.** Consider the Mealy machines $\mathcal{S}$ of Figure 6, which has a single state, two inputs a and b, and minimal state cover $A = \{\epsilon\}$. Test suite $T = \{ab\}$ does not meet the

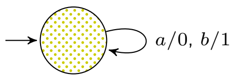

Figure 6 Test suite $T = \{ab\}$ is 0-$\{\epsilon\}$-complete for the above specification.

conditions of Corollary 4.10, since the basis of the corresponding testing tree is not complete. Nevertheless, we claim that T is 0-A-complete. Since A is a singleton, the fault domain $\mathcal{U}^A$ is empty. The fault domain $\mathcal{U}_0^A$ only contains Mealy machines with a single state, and self-loop transitions for inputs a and b. Test suite T verifies that the outputs of these transitions are in agreement with $\mathcal{S}$. Thus, any machine in $\mathcal{U}_0^A$ that passes T will be equivalent to $\mathcal{S}$. Hence T is 0-A-complete.

Theorem 4.9 and Corollary 4.10 do not require that all states in $F^{<k}$ are identified, but the other conditions of the theorem/corollary already imply this.

▶ **Proposition 4.13.** *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 A contains an example to illustrate that the converse implication does not hold: even if all states in $B \cup F^{\leq k}$ are identified, condition (1) may not hold. The next proposition asserts that 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 4.14.** *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 $\forall i, j\ \forall q \in F^i\ \forall r \in F^j : 0 \leq i < j \leq k \quad \Rightarrow \quad C(q) = C(r) \vee q \# r$*

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

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

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 [33, 11].

▶ **Proposition 4.15.** *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*

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

4.5 Algorithm

We will now present an algorithm 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. In practice, models of realistic protocols often have up to a few dozen states and inputs [24]. This means that if k equals 2 or 3, the observation tree will contain up to a few thousand states, and so with some optimization (e.g. on the fly computation of apartness pairs) our algorithm is practical. For larger benchmarks the observation trees already contain millions of states, and our algorithm cannot be applied. Nevertheless, from a theoretical perspective it is interesting to know that the apartness relation (and hence also our sufficient condition for k-A-completeness) can be computed in polynomial time.

Algorithm 1 Computing the apartness relation.

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

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 4.16.** *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 $q, q' \in Q^{\mathcal{T}}$.*

Once we know the apartness relation, there are several things we can do, e.g., we may check in $\Theta(N^2)$ time whether the conditions of Theorem 4.9 hold (but note that N grows exponentially in k). First we check in linear time whether all states in B and $F^{<k}$ are complete. Next, in one pass over array *Apart*, we compute the candidate sets for all frontier states. By Proposition 4.13, if some state in the frontier has a candidate set with more than one element, then the conditions of Theorem 4.9 do not hold. Otherwise, we can check in constant time whether frontier states have the same candidate set. Finally, we check condition (1) with a double for-loop over F^k and $F^{<k}$.

If we can compute the apartness relation, we may also select appropriate state identifiers on-the-fly in order to generate small k-A-complete test suites (similar to the H-method [6]), and we can 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 4.10. We also present an alternative m-completeness proof of the H-method [6], which is a minor variation of the proof of Theorem 4.9. In order to define the Wp and HSI methods, we need certain sets (of sets) of sequences. The following definitions are based on [22]; see [5, 22] 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, i.e., a witness for their apartness.*
- *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.

k-A-completeness of the Wp-method [9] follows from Corollary 4.10 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 [19] and Petrenko et al [41, 28] follows from Corollary 4.10 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 [39, 3], and the UIOv-method of [2] are instances of the Wp-method, and the ADS-method of [17] and the hybrid ADS method of [31] are instances of the HSI-method. This means that 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 4.9 which requires that all states in $F^{\leq k}$ are identified and replaces condition (1) by

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

By Proposition 4.14, condition (3) is implied by condition (1) of Theorem 4.9. Appendix B contains 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. Below we present an alternative formulation of the m-completeness result for the H-method of [6], restated for our setting. The full version of this article contains a proof of this proposition that uses the same proof technique as Theorem 4.9.

▶ **Proposition 5.4** (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. Then T is m-complete.*

6 Conclusions and Future Work

In order to solve a long-standing open problem of Hennie [12], we proposed the notion of k-A-completeness. We showed that the fault domain for k-A-completeness is larger than the one for m-completeness (if $m = |A| + k$), and includes FSMs with a number of extra states that grows exponentially in k. We provided a sufficient condition for k-A-completeness in terms of apartness of states of the observation 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 [9] and the HSI-method of [19, 41, 28]. 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 [30], the H-method [6], and the SPYH-method [32] are not k-A-complete.

We view our results as an important step towards the definition of fault domains that capture realistic assumptions about possible faults, but still allow for the design of sufficiently small test suites that are complete for the fault domain and can be run within reasonable time. A promising research direction is to reduce the size of the fault domains via reasonable assumptions on the structure of the implementation under test. Kruger et al [16] show that significantly smaller test suites suffice if the fault domains $\mathcal{U}_m$ are reduced by making plausible assumptions about the implementation. The same approach can also be applied for the fault domains $\mathcal{U}_k^A \cup \mathcal{U}^A$ proposed in this article. We expect that our results will find applications in the area of model learning [1, 25, 34, 13]. Black-box conformance testing, which is used to find counterexamples for hypothesis models, has become the main bottleneck in applications of model learning [34, 40]. This provides motivation and urgency to find fault domains that provide a better fit with applications and allow for smaller test suites [16].

An important question for future research is to explore empirical evidence for the usefulness of our new fault domains: how often are faulty implementations of real systems contained in fault domains $\mathcal{U}_k^A$, for small k and a sensible choice of A? Hübner et al [14] investigated how test generation methods perform for SUTs whose behaviors lie outside the fault domain.

They considered some realistic SUTs – SystemC implementations of a ceiling speed monitor and an airbag controller described in SysML – 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 [4, 30] 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 [28].

Closest to our characterization is the work of Sachtleben [29], 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 [29] 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 [29] can be adapted to our notion of k-A-completeness.

An obvious direction for future work is to extend our notion of k-A-completeness to richer classes of models, such as timed automata and register automata. The recent work of [15] may provide some guidance here. It will also be interesting to explore if our characterization can be used for efficient test suite generation, or for pruning test suites that have been generated by other methods.

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.
2. W. Y. L. Chan, C. T. Vuong, and M. R. Otp. An improved protocol test generation procedure based on UIOs. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols, September 19-22, 1989*, SIGCOMM '89, pages 283–294, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/75246.75274.
3. T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978. doi:10.1109/TSE.1978.231496.
4. Adenilso da Silva Simão, Alexandre Petrenko, and Nina Yevtushenko. Generating reduced tests for FSMs with extra states. In Manuel Núñez, Paul Baker, and Mercedes G. Merayo, editors, *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*, volume 5826 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2009. doi:10.1007/978-3-642-05031-2_9.
5. Rita Dorofeeva, Khaled El-Fakih, Stéphane Maag, Ana R. Cavalli, and Nina Yevtushenko. FSM-based conformance testing methods: A survey annotated with experimental evaluation. *Information & Software Technology*, 52(12):1286–1297, 2010. doi:10.1016/j.infsof.2010.07.001.

6 Rita Dorofeeva, Khaled El-Fakih, and Nina Yevtushenko. An improved conformance testing method. In Farn Wang, editor, *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings*, volume 3731 of *Lecture Notes in Computer Science*, pages 204–218. Springer, 2005. `doi:10.1007/11562436_16`.

7 Feodor F. Dragan and Arne Leitert. On the minimum eccentricity shortest path problem. *Theoretical Computer Science*, 694:66–78, 2017. `doi:10.1016/j.tcs.2017.07.004`.

8 P. Fiterău-Broştean, R. Janssen, and F.W. Vaandrager. Combining model learning and model checking to analyze TCP implementations. In S. Chaudhuri and A. Farzan, editors, *Proceedings 28th International Conference on Computer Aided Verification (CAV'16),* Toronto, Ontario, Canada, volume 9780 of *Lecture Notes in Computer Science*, pages 454–471. Springer, 2016. `doi:10.1007/978-3-319-41540-6_25`.

9 S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991. `doi:10.1109/32.87284`.

10 Maciej Gazda and Robert M. Hierons. Model independent refusal trace testing. *Science of Computer Programming*, 239:103173, 2025. `doi:10.1016/j.scico.2024.103173`.

11 Herman Geuvers and Bart Jacobs. Relating apartness and bisimulation. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. `doi:10.46298/lmcs-17(3:15)2021`.

12 F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, 1964. `doi:10.1109/SWCT.1964.8`.

13 Falk Howar and Bernhard Steffen. Active automata learning in practice. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers*, pages 123–148. Springer International Publishing, 2018.

14 Felix Hübner, Wen-ling Huang, and Jan Peleska. Experimental evaluation of a novel equivalence class partition testing strategy. *Softw. Syst. Model.*, 18(1):423–443, 2019. `doi:10.1007/S10270-017-0595-8`.

15 Bálint Kocsis and Jurriaan Rot. Complete test suites for automata in monoidal closed categories. In Parosh Aziz Abdulla and Delia Kesner, editors, *Foundations of Software Science and Computation Structures - 28th International Conference, FoSSaCS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings*, volume 15691 of *Lecture Notes in Computer Science*, pages 198–219. Springer, 2025. `doi:10.1007/978-3-031-90897-2_10`.

16 Loes Kruger, Sebastian Junges, and Jurriaan Rot. Small test suites for active automata learning. In Bernd Finkbeiner and Laura Kovács, editors, *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*, volume 14571 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2024. `doi:10.1007/978-3-031-57249-4_6`.

17 D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994. `doi:10.1109/12.272431`.

18 D. Lee and M. Yannakakis. Principles and methods of testing finite state machines — a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996. `doi:10.1109/5.533956`.

19 Gang Luo, Alexandre Petrenko, and Gregor von Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In Tadanori Mizuno, Teruo Higashino, and Norio Shiratori, editors, *Protocol Test Systems: 7th workshop 7th IFIP WG 6.1 international workshop on protocol text systems*, pages 95–110, Boston, MA, 1995. Springer US. `doi:10.1007/978-0-387-34883-4_6`.

20 T. Maarse. *Active Mealy Machine Learning using Action Refinements.* Master thesis, Radboud University Nijmegen, August 2020. URL: `https://www.cs.ru.nl/F.Vaandrager/thesis-2019_NWI-IMC048_s4416295.pdf`.

21 I. Melse. *Sufficient Conditions for k-Completeness Using Observation Trees and Apartness.* Bachelor thesis, ICIS, Radboud University Nijmegen, April 2024. URL: `https://www.cs.ru.nl/bachelors-theses/2024/Ivo_Melse___s1088677___Sufficient_Conditions_for_k-completeness_using_Observation_Trees_and_Apartness.pdf`.

22 J. Moerman. *Nominal Techniques and Black Box Testing for Automata Learning.* PhD thesis, Radboud University Nijmegen, July 2019.

23 E.F. Moore. Gedanken-experiments on sequential machines. In *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 129–153. Princeton University Press, 1956.

24 Daniel Neider, Rick Smetsers, Frits W. Vaandrager, and Harco Kuppens. Benchmarks for automata learning and conformance testing. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not?*, volume 11200 of *Lecture Notes in Computer Science*, pages 390–416. Springer, 2018. `doi:10.1007/978-3-030-22348-9_23`.

25 Doron Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Proceedings FORTE*, volume 156 of *IFIP Conference Proceedings*, pages 225–240. Kluwer, 1999.

26 A. Petrenko, T. Higashino, and T. Kaji. Handling redundant and additional states in protocol testing. In A. Cavalli and S. Budkowski, editors, *Proceedings of the 8th International Workshop on Protocol Test Systems IWPTS '95,* Paris, France, pages 307–322, 1995.

27 Alexandre Petrenko and Nina Yevtushenko. Conformance tests as checking experiments for partial nondeterministic FSM. In Wolfgang Grieskamp and Carsten Weise, editors, *Formal Approaches to Software Testing, 5th International Workshop, FATES 2005, Edinburgh, UK, July 11, 2005, Revised Selected Papers*, volume 3997 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2005. `doi:10.1007/11759744_9`.

28 Alexandre Petrenko, Nina Yevtushenko, Alexandre Lebedev, and Anindya Das. Nondeterministic state machines in protocol conformance testing. In Omar Rafiq, editor, *Protocol Test Systems, VI, Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems, Pau, France, 28-30 September, 1993*, volume C-19 of *IFIP Transactions*, pages 363–378. North-Holland, 1993.

29 Robert Sachtleben. Unifying frameworks for complete test strategies. *Science of Computer Programming*, 237:103135, 2024. `doi:10.1016/j.scico.2024.103135`.

30 Adenilso Simão, Alexandre Petrenko, and Nina Yevtushenko. On reducing test length for FSMs with extra states. *Softw. Test. Verification Reliab.*, 22(6):435–454, 2012. `doi:10.1002/STVR.452`.

31 Wouter Smeenk, Joshua Moerman, Frits W. Vaandrager, and David N. Jansen. Applying automata learning to embedded control software. In Michael J. Butler, Sylvain Conchon, and Fatiha Zaïdi, editors, *Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, France, 2015, Proceedings*, volume 9407 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2015. `doi:10.1007/978-3-319-25423-4_5`.

32 Michal Soucha and Kirill Bogdanov. 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*, pages 194–203. IEEE Computer Society, 2018. `doi:10.1109/ICSTW.2018.00050`.

33 A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2 edition, 2000. `doi:10.1017/CBO9781139168717`.

34 Frits Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, February 2017. `doi:10.1145/2967606`.

35 Frits Vaandrager. A new perspective on conformance testing based on apartness. In Venanzio Capretta, Robbert Krebbers, and Freek Wiedijk, editors, *Logics and Type Systems in Theory and Practice: Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday*, pages 225–240, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-61716-4_15.

36 Frits W. Vaandrager, Paul Fiterau-Brostean, and Ivo Melse. Completeness of FSM test suites reconsidered. *CoRR*, abs/2410.19405, 2024. doi:10.48550/arXiv.2410.19405.

37 Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022. doi:10.1007/978-3-030-99524-9_12.

38 Petra van den Bos, Ramon Janssen, and Joshua Moerman. n-Complete test suites for IOCO. *Softw. Qual. J.*, 27(2):563–588, 2019. doi:10.1007/S11219-018-9422-X.

39 M.P. Vasilevskii. Failure diagnosis of automata. *Cybernetics and System Analysis*, 9(4):653–665, 1973. (Translated from Kibernetika, No. 4, pp. 98-108, July-August, 1973.). doi:10.1007/BF01068590.

40 Nan Yang, Kousar Aslam, Ramon R. H. Schiffelers, Leonard Lensink, Dennis Hendriks, Loek Cleophas, and Alexander Serebrenik. Improving model inference in industry by combining active and passive learning. In Xinyu Wang, David Lo, and Emad Shihab, editors, *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, pages 253–263. IEEE, 2019. doi:10.1109/SANER.2019.8668007.

41 N. V. Yevtushenko and A. F. Petrenko. Synthesis of test experiments in some classes of automata. *Autom. Control Comput. Sci.*, 24(4):50–55, April 1991.

A Condition (1) is Needed

Proposition 4.13 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 7 present a counterexample with $k = 1$ and $A = \{\epsilon, l, r\}$.

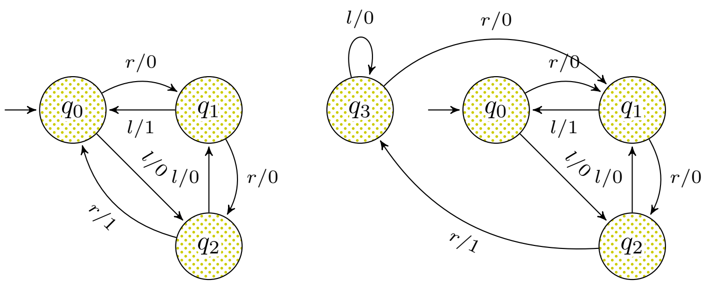

Figure 7 A specification $\mathcal{S}$ (left) and a faulty implementation $\mathcal{M}$ (right).

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 8 shows an observation tree $\mathcal{T}$ for both $\mathcal{S}$ and $\mathcal{M}$.
Observation tree $\mathcal{T}$ meets all the requirements of Theorem 4.9, 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 complete, and all states in B and $F^{\leq 1}$ are identified. However, $\mathcal{T}$ 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 4.9 doesn't hold.

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

▶ **Example B.1.** Figure 9(left) shows the running example $\mathcal{S}$ from the article by Simão, Petrenko and Yevtushenko that introduces the SPY-method [30]. Using this method, a 3-complete test suite $\{aaaa, baababba, bbabaa\}$ was derived in [30]. Consider the minimal state cover $A = \{\epsilon, a\}$ for $\mathcal{S}$. Implementation $\mathcal{M}$ from Figure 9(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 [30] is not 1-A-complete.

▶ **Example B.2.** Figure 11 shows the tree for a 3-complete test suite generated by the H-method of Dorofeeva et al [6] for the machine of Figure 10(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 10(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.

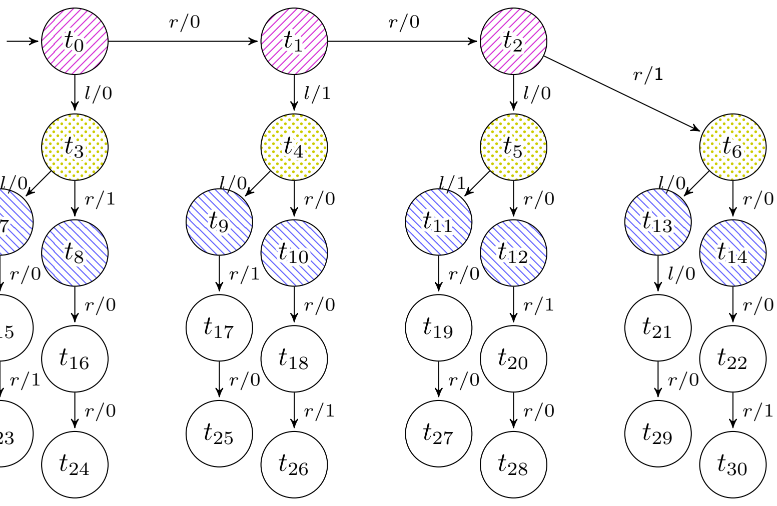

Figure 8 Observation tree for FSMs $\mathcal{S}$ and $\mathcal{M}$ from Figure 7.

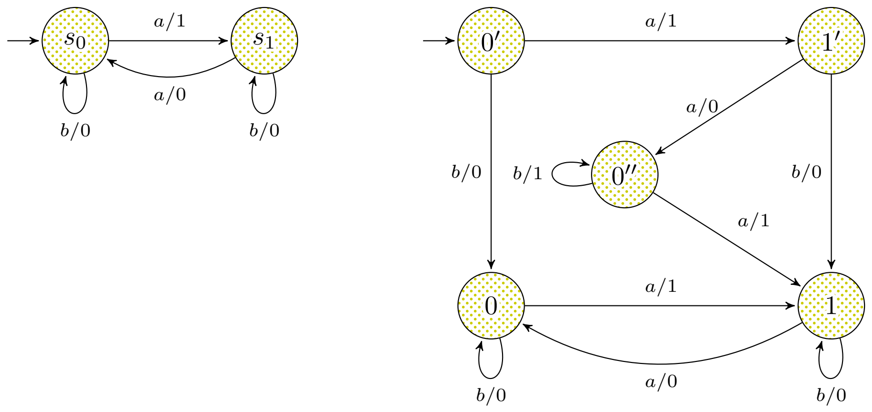

Figure 9 An implementation from fault domain $\mathcal{U}_1^A$ (right) that passes the 3-complete test suite $\{aaaa, baababba, bbabaa\}$ that was constructed for the specification (left) using the SPY-method.

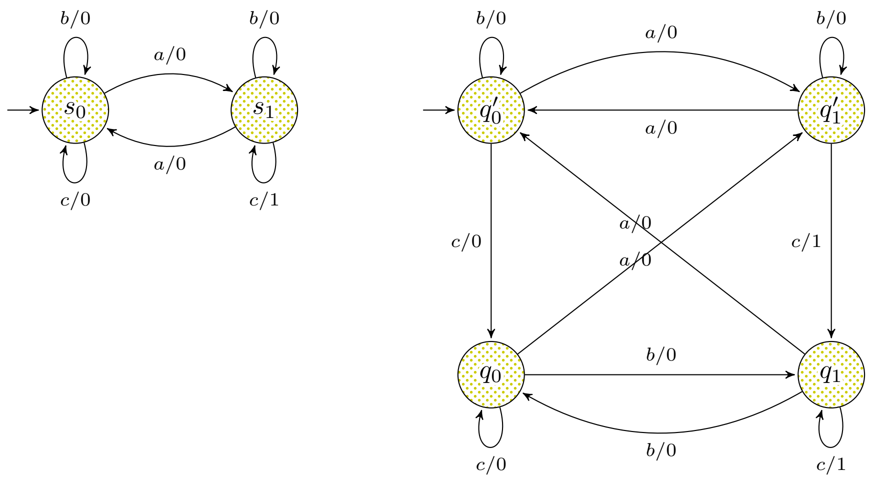

Figure 10 Specification (left) and implementation (right) from fault domain $\mathcal{U}_1^A$ that passes the test suite of Figure 11.

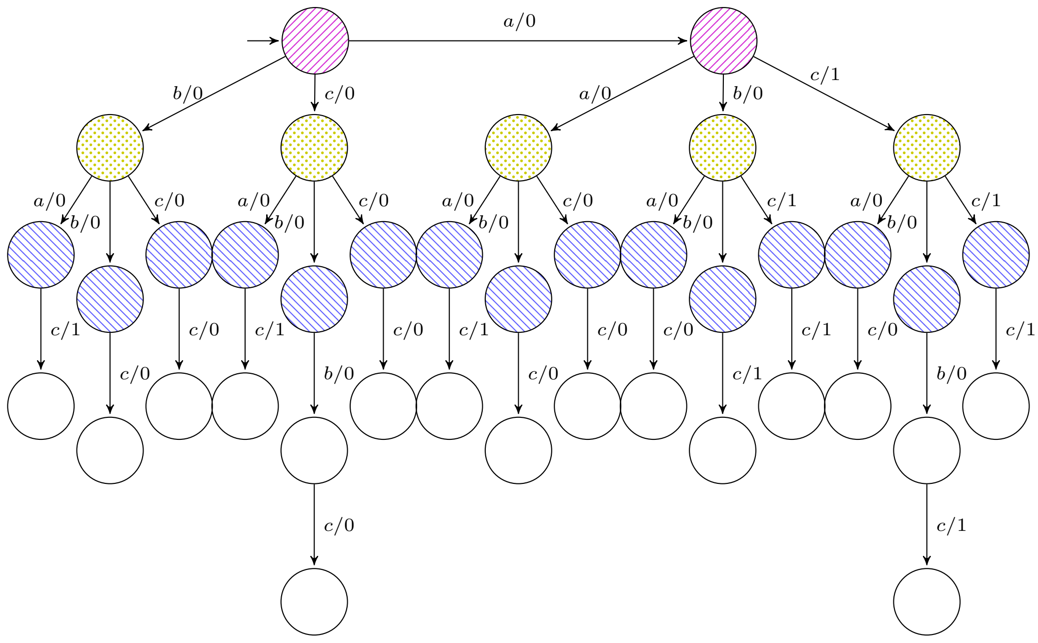

Figure 11 Testing tree for 3-complete test suite constructed for the specification of Figure 10(left) using the H-method.