A New Perspective on Conformance Testing Based on Apartness[⋆]

Frits Vaandrager[✉]

Institute for Computing and Information Sciences,
Radboud University, Nijmegen, the Netherlands
f.vaandrager@cs.ru.nl

Abstract. We revisit the classic problem of black box conformance testing and present a generalization of the k-completeness result of Vasilevskii and Chow, phrased entirely in terms of properties of the observation/prefix tree induced by a test suite, in particular in terms of apartness relations between states. The original result of Vasilevskii and Chow is then a corollary of our result. Also k-completeness results for other test methods that have been proposed in the literature, such as the Wp-method and the HSI-method, follow from our characterization. Based on the apartness relations in the observation tree, we may determine whether some test is redundant or can be shortened.

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

1 Introduction

In this note, we revisit the classic problem of black box conformance testing. We consider the simple setting in which both the specification and the black box implementation can be described as (deterministic, complete) finite state machines (a.k.a. Mealy machines). In this setting, we refer to a sequence of inputs σ as a *test*. Given a specification $\mathcal{S}$, we say that an implementation $\mathcal{M}$ *passes* test σ if σ triggers the same outputs in $\mathcal{M}$ and $\mathcal{S}$. Otherwise, we say that $\mathcal{M}$ *fails* test σ. Implementation $\mathcal{M}$ *conforms* to specification $\mathcal{S}$ if it passes all tests. In our setting this means that $\mathcal{M}$ and $\mathcal{S}$ have equivalent behavior. The task of a tester of a black box implementation $\mathcal{M}$ is to find a test σ (if it exists) such that $\mathcal{M}$ fails σ. Figure 1 shows an example (taken from [11]). Here the test $\sigma = aba$ triggers outputs 010 in the specification, but outputs 011 in the implementation. Thus, $\mathcal{M}$ fails test σ and implementation $\mathcal{M}$ does not conform to specification $\mathcal{S}$.

Ideally, given a specification $\mathcal{S}$, a tester would like to compute a finite set of tests T, called a *test suite*, that is complete in the sense that, for any implementation $\mathcal{M}$, $\mathcal{M}$ will pass all tests in T if and only if $\mathcal{M}$ conforms to S. Unfortunately, such a test suite does not exist: for any finite test suite T the number of inputs

⋆ Research supported by NWO TOP project 612.001.852 “Grey-box learning of Interfaces for Refactoring Legacy Software (GIRLS)”.

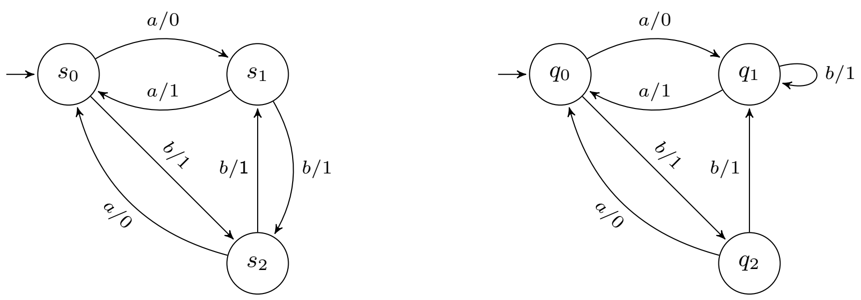

Fig. 1: A specification $\mathcal{S}$ (left) and a faulty implementation $\mathcal{M}$ (right).

in tests will be bounded by some number n. Thus, for any specification $\mathcal{S}$ and any n we may construct an implementation $\mathcal{M}$ that behaves exactly like $\mathcal{S}$ for the first n inputs, but behaves differently from that point onwards. Even though this $\mathcal{M}$ does not conform to $\mathcal{S}$, it will pass all tests in T.

In two classic papers, Vasilevskii [20] and Chow [3] independently showed that, for any specification $\mathcal{S}$ and any natural number k, a finite test suite T can be constructed that is *k-complete* in the sense that, for any implementation $\mathcal{M}$ with at most k more states than $\mathcal{S}$, $\mathcal{M}$ passes T if and only if $\mathcal{M}$ conforms to $\mathcal{S}$. The so-called *W-method* proposed by Vasilevskii [20] and Chow [3] for constructing a k-complete test suite is actually quite simple:

– First, construct a *state cover* for $\mathcal{S}$: a finite, prefix closed set of input sequences A such that every state of $\mathcal{S}$ is reached by some sequence in A. For example, the set $A = \{\epsilon, a, b\}$ is a state cover for the specification $\mathcal{S}$ from Figure 1, since s_0 is reached by the empty sequence ϵ, s_1 is reached by a, and s_2 is reached by b.
– Next, construct a *characterization set* for $\mathcal{S}$: a nonempty, finite set of input sequences W, such that for each pair of inequivalent states s and t, W contains a separating sequence for s and t. Here a *separating sequence* for s and t is a sequence σ such that when σ is applied in s the resulting outputs are different from those when σ is applied in t. Two states s and t of $\mathcal{S}$ are *equivalent* if there exists no separating sequence for them. For example, aa is a separating sequence for all pairs of distinct states of $\mathcal{S}$ since in s_0 it triggers outputs 01, in s_1 it triggers outputs 10, and in s_2 it triggers outputs 00. Thus $W = \{aa\}$ is a characterization set for $\mathcal{S}$.
– Finally, a k-complete test suite T is defined by $T = A \cdot I^{\leq k+1} \cdot W$, where $I^{\leq k+1}$ is the set of all input sequences with length less than or equal to $k+1$, and "$\cdot$" denotes concatenation of sequences, extended to sets of sequences by pointwise extension.

Intuitively, in the simple case with $k = 0$, input sequences $A \cdot W$ test whether all states of the specification are present in the implementation, and input sequences $A \cdot I \cdot W$ test whether all transitions produce the correct output and lead to the correct target state. If we apply the W-method to construct a 0-complete test suite for the specification of Figure 1, we obtain $T = \{aa, aaa, baa, aaaa, abaa, baaa, bbaa\}$. We can always omit tests that are a prefix of another test: if an implementation fails on a prefix of a test then it will also fail on the full test. Thus we can omit tests aa, aaa and baa, and obtain a 0-complete test suite with 4 tests. Indeed, the implementation from Figure 1 fails the test $abaa$.

Numerous variations and improvements of the W-method have been proposed in the literature; we refer to [9,4,11] for overviews and further references. Still, for all these methods it is unclear how to (1) establish whether certain tests are redundant or can be shortened (for instance, in the example test suite T, $abaa$ can be replaced by the shorter aba without compromising 0-completeness), (2) compute which tests should be selected first in order to maximize the likelihood that bugs will be discovered, and related (3) quantify how performing some test will reduce our uncertainty whether an implementation conforms to a specification.

In order to be able address these fundamental questions, we present a new perspective on conformance testing, drawing inspiration from the recently developed $L^{\#}$ learning algorithm [19]. Actually, given the natural duality between learning and testing, first observed by Weyuker [21], it is not surprising that ideas regarding active learning of finite state machines can be applied in the setting of conformance testing. A first idea from the $L^{\#}$ algorithm that we will use is to store the outcomes of all experiments/tests in a single data structure, the *observation tree* (a.k.a. prefix tree). This allows us, for instance, to base the choice of the next test on the set of tests executed thus far. A second idea that we adapt from $L^{\#}$ is the concept of *apartness*, a constructive form of inequality [18]. The notion of apartness is standard in constructive real analysis and goes back to Brouwer, with Heyting giving an axiomatic treatment in [7]. The importance of apartness for automata theory and concurrency theory was first observed by Herman Geuvers and Bart Jacobs [6]. This insight was an inspiration for the work of [19] and for the present note.

The main result that we present here is a generalization of the k-completeness result of [20,3], phrased entirely in terms of properties of the observation tree, in particular in terms of apartness relations between states. The original result of [20,3] is then a corollary of our result. Also k-completeness results for other test methods that have been proposed in the literature follow from our characterization. Based on the apartness relations in the observation tree, we may determine whether a certain test is redundant or can be shortened.

The rest of this note is structured as follows. First we recall the formal definitions of (partial) Mealy machines, observation trees and apartness in Section 2. Next, we present our main result in Section 3. The connections with existing k-completeness results are discussed in Section 4. In Section 5, we discuss implications of our results and directions for future research.

2 Partial Mealy Machines and Apartness

In this preliminary section, which is largely copied from [19] (but with the word "learning" replaced by "testing"), we formalize some of the key concepts that play a role in our result: Mealy machines, observation trees and apartness. Since observation trees are (a specific type of) partial Mealy machines, we need to fix some notation for partial maps.

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 $\{(x,y) \in X \times Y \mid f(x) = y\}$. There is a partial order on $X \rightharpoonup Y$ defined by $f \sqsubseteq g$ for $f, g\colon X \rightharpoonup Y$ iff for all $x \in X$, $f(x){\downarrow}$ implies $g(x){\downarrow}$ and $f(x) = g(x)$.

Throughout this paper, we fix a nonempty, finite set I of *inputs* and a set O of *outputs*.

Definition 2.1. *A **Mealy machine** is a tuple $\mathcal{M} = (Q, q_0, \delta, \lambda)$, where*
- *Q is a finite set of **states** and $q_0 \in Q$ is the **initial state**,*
- *$\langle \lambda, \delta \rangle\colon Q \times I \rightharpoonup O \times Q$ is a partial map whose components are an **output function** $\lambda\colon Q \times I \rightharpoonup O$ and a **transition function** $\delta\colon Q \times I \rightharpoonup Q$.*

*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 $\mathcal{M}$ **complete** iff δ is total, i.e., $\delta(q,i)$ is defined for all states q and inputs i. We generalize the transition and output functions to input words of length $n \in \mathbb{N}$ by composing $\langle \lambda, \delta \rangle$ n times with itself: we define maps $\langle \lambda_n, \delta_n \rangle\colon Q \times I^n \rightharpoonup O^n \times Q$ by $\langle \lambda_0, \delta_0 \rangle = \mathsf{id}_Q$ and*

$$\langle \lambda_{n+1}, \delta_{n+1} \rangle\colon \quad Q \times I^{n+1} \xrightharpoonup{\langle \lambda_n, \delta_n \rangle \times \mathsf{id}_I} O^n \times Q \times I \xrightharpoonup{\mathsf{id}_{O^n} \times \langle \lambda, \delta \rangle} O^{n+1} \times Q$$

Whenever it is clear from the context, we use λ and δ also for words.

Definition 2.2 (Equivalence and minimality). *The semantics of a state q is a map $[\![q]\!]\colon I^* \rightharpoonup O^*$ defined by $[\![q]\!](\sigma) = \lambda(q,\sigma)$. States q, q' in possibly different Mealy machines are **equivalent**, written $q \approx q'$, iff $[\![q]\!] = [\![q']\!]$. Mealy machines $\mathcal{M}$ and $\mathcal{N}$ are **equivalent** iff their respective initial states are equivalent: $q_0^{\mathcal{M}} \approx q_0^{\mathcal{N}}$. Mealy machine $\mathcal{M}$ is **minimal** iff, for all pairs of states q, q', $q \approx q'$ iff $q = q'$.*

In our testing setting, an *undefined* value in the partial transition map represents lack of knowledge. We consider maps between Mealy machines that preserve existing transitions, but possibly extend the knowledge of transitions:

Definition 2.3 (Simulation). *For Mealy machines $\mathcal{M}$ and $\mathcal{N}$, a **functional simulation** $f\colon \mathcal{M} \to \mathcal{N}$ is a map $f\colon Q^{\mathcal{M}} \to Q^{\mathcal{N}}$ with*

$$f(q_0^{\mathcal{M}}) = q_0^{\mathcal{N}} \qquad \text{and} \qquad q \xrightarrow{i/o} q' \text{ implies } f(q) \xrightarrow{i/o} f(q').$$

Intuitively, a functional simulation preserves transitions and the initial state.

Lemma 2.4. *For a functional simulation $f\colon \mathcal{M} \to \mathcal{N}$ and $q \in Q^{\mathcal{M}}$, we have $[\![q]\!] \sqsubseteq [\![f(q)]\!]$.*

For a given machine $\mathcal{M}$, an observation tree is simply a Mealy machine itself which represents the inputs and outputs we have observed so far during testing. 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 sequence $\sigma \in I^*$ s.t. $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) = q$. We write* $\mathsf{access}(q)$ *for the sequence of inputs leading to q. A tree $\mathcal{T}$ is an **observation tree** for a Mealy machine $\mathcal{M}$ iff there is a functional simulation $f\colon \mathcal{T} \to \mathcal{M}$.*

Figure 2 shows an observation tree for the Mealy machine displayed on the right. The functional simulation f is indicated via state colors. By performing

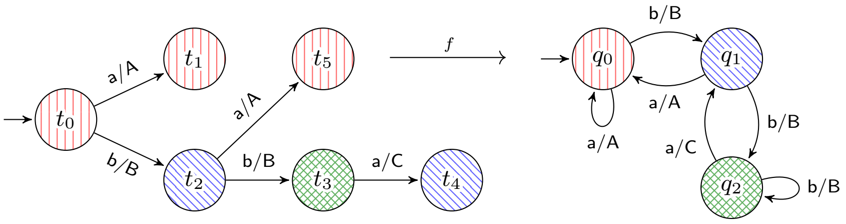

Fig. 2: An observation tree (left) for a Mealy machine (right).

tests, a tester may construct an observation tree $\mathcal{T}$ for implementation $\mathcal{M}$. The observation tree of Figure 2, for instance, may be constructed by performing tests a, ba, and bba. There is a one-to-one correspondence between observation trees (up to isomorphism) and finite test suites T that do not contain redundant prefixes (i.e., if $\sigma, \rho \in T$ with σ a prefix of ρ then we require $\sigma = \rho$.)

Suppose $\mathcal{T}$ is an observation tree for implementation $\mathcal{M}$. If all tests, as recorded in the tree, have passed then $\mathcal{T}$ is also an observation tree for $\mathcal{S}$. However, as soon as a test fails then $\mathcal{T}$ is no longer an observation tree for $\mathcal{S}$: there exists no functional simulation from $\mathcal{T}$ to $\mathcal{S}$, since the observation tree contains a state q such that the output in response to input sequence $\mathsf{access}(q)$ is different for $\mathcal{S}$ and $\mathcal{M}$.

Even though the tester constructs the observation tree $\mathcal{T}$, they do not know the functional simulation. However, by analysis of the observation tree, a tester may infer that certain states in the tree cannot have the same color, that is, they cannot be mapped to same states of $\mathcal{M}$ by a functional simulation. In this analysis, the concept of *apartness*, a constructive form of inequality, plays a crucial role [18,6].

Definition 2.6 (Apartness). *For a Mealy machine $\mathcal{M}$, we say that states $q, p \in Q^{\mathcal{M}}$ are **apart** (written $q \# p$) iff there is some $\sigma \in I^*$ such that $[\![q]\!](\sigma)\downarrow$,*

$[\![p]\!](\sigma)\downarrow$, and $[\![q]\!](\sigma) \neq [\![p]\!](\sigma)$. We say that σ is the ***witness*** *of $q \# p$ and write $\sigma \vdash q \# p$.*

Note that the apartness relation $\# \subseteq Q \times Q$ is irreflexive and symmetric. Within conformance testing theory, a witness is commonly called *separating sequence* [17]. For the observation tree of Figure 2 we may derive the following apartness pairs and corresponding witnesses:

$$a \vdash t_0 \# t_3 \qquad a \vdash t_2 \# t_3 \qquad b\, a \vdash t_0 \# t_2$$

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

Lemma 2.7. *For a functional simulation $f \colon \mathcal{T} \to \mathcal{M}$,*

$$q \# p \text{ in } \mathcal{T} \qquad \Longrightarrow \qquad f(q) \not\approx f(p) \text{ in } \mathcal{M} \qquad \text{for all } q, p \in Q^{\mathcal{T}}.$$

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

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

A tester may use the weak co-transitivity property during testing. For instance in Fig. 2, by performing the test aba, consisting of the access sequence for t_1 concatenated with the witness ba for $t_0 \# t_2$, co-transitivity ensures that $t_0 \# t_1$ or $t_2 \# t_1$. By inspecting the outputs, a tester may conclude that $t_2 \# t_1$.

3 Main Result

In this section, we describe a sufficient condition for a test suite to be k-complete, phrased in terms of the corresponding observation 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 certain apartness relations between states of the tree should hold.

In order to present our condition and the proof of its correctness, we first need to introduce some auxiliary termininology.

Definition 3.1 (Stratification). *Let $A \subseteq I^*$ be a nonempty, finite, prefix closed set of input sequences, and let $\mathcal{T}$ be an observation tree. Then A induces a* ***stratification*** *of $Q^{\mathcal{T}}$ as follows:*

1. *A state q of $\mathcal{T}$ is called a* ***basis state*** *iff* $\mathsf{access}(q) \in A$. *We write B to denote the set of basis states: $B = \{q \in Q^{\mathcal{T}} \mid \mathsf{access}(q) \in A\}$. Note that, since A is nonempty and prefix closed, initial state $q_0^{\mathcal{T}}$ is in the basis, and all states on the path leading to a basis state are basis states as well.*
2. *We write F^0 for the set of immediate successors of basis states that are not basis states themselves: $F^0 := \{q' \in Q^{\mathcal{T}} \setminus B \mid \exists q \in B, i \in I : q' = \delta^{\mathcal{T}}(q,i)\}$. We refer to F^0 as the* ***0-level frontier****.*
3. *For $k > 0$, the* ***k-level frontier*** *F^k is the set of immediate successors of $k-1$-level frontier states: $F^k := \{q' \in Q^{\mathcal{T}} \mid \exists q \in F^{k-1}, i \in I : q' = \delta^{\mathcal{T}}(q,i)\}$.*

We say that basis B is ***complete*** *if for each $\sigma \in A$ there is a state $q \in B$ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) = q$. The 0-level frontier is* ***complete*** *if for each $q \in B$ and for each $i \in I$, $\delta^{\mathcal{T}}(q,i)\downarrow$. For $k > 0$, the k-level frontier is* ***complete*** *if for each $q \in F^{k-1}$ and for each $i \in I$, $\delta^{\mathcal{T}}(q,i)\downarrow$.*

For each state q of an observation tree, we define the ***candidate set*** *$C(q)$ as the set of basis states that are not apart from q: $C(q) = \{q' \in B \mid \neg(q \# q')\}$. A state q of an observation tree is* ***identified*** *if its candidate set is a singleton.*

Example 3.2. Figure 3 shows the stratification for an observation tree for specification $\mathcal{S}$ from Figure 1 induced by $A = \{\epsilon, a, b\}$. States from sets B, F^0, F^1 and F^2 are marked with different colors. In Figure 3, B and F^0 are complete,

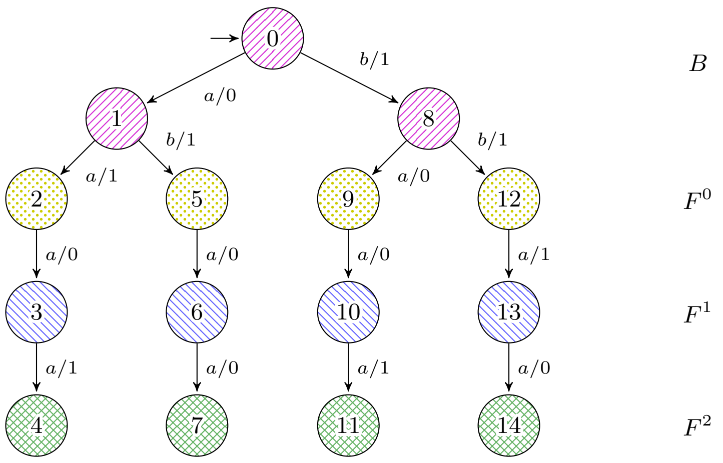

Fig. 3: Stratification of an observation tree induced by $A = \{\epsilon, a, b\}$.

but F^1 and F^2 are incomplete (since states of F^0 and F^1 have no outgoing b-transitions). Witness aa shows that the three basis states are pairwise apart, and therefore identified. 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 $C(4) = C(7) = C(11) = C(14) = \{0, 1, 8\}$.

In the proof of our main result, we use a bisimulation relation and the well-known fact that for (deterministic) Mealy machines bisimulation equivalence coincides with the standard behavioral equivalence for Mealy machines from Definition 2.2.

Definition 3.3 (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 $q \in Q^{\mathcal{M}}$, $r \in Q^{\mathcal{N}}$, $i \in I$, $o \in O$,*

$$q_0^{\mathcal{M}} \; R \; q_0^{\mathcal{N}} \qquad \textit{and} \qquad q \; R \; r \;\wedge\; q \xrightarrow{i/o} q' \;\Rightarrow\; \exists r' \colon r \xrightarrow{i/o} r' \;\wedge\; q' \; R \; r'$$

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 [13], is easy to prove.

Lemma 3.4. *Let $\mathcal{M}$ and $\mathcal{N}$ be complete Mealy machines. Then $\mathcal{M} \simeq \mathcal{N}$ iff $\mathcal{M} \approx \mathcal{N}$.*

We are now prepared to state and prove our main result.

Theorem 3.5. *Let $\mathcal{M}$ and $\mathcal{S}$ be complete Mealy machines and let $\mathcal{T}$ be an observation tree for both $\mathcal{M}$ and $\mathcal{S}$. Let $A \subseteq I^*$ be a state cover for $\mathcal{S}$, and let $B, F^0, F^1, \ldots$ be the stratification of $Q^{\mathcal{T}}$ induced by A. Let $k \geq 0$. Suppose that B and $F^0, \ldots, F^k$ are all complete, all states in B and $F^0, \ldots, F^k$ are identified, and the following co-transitivity property holds*

$$\forall r \in B \; \forall t' \in F^k \; \forall t'' \in F^0 \cup \cdots \cup F^{k-1} \;\; : \;\; r \;\#\; t' \implies r \;\#\; t'' \vee t' \;\#\; t'' \quad (1)$$

Suppose $\mathcal{M}$ has at most k more states than $\mathcal{S}$. Then $\mathcal{S} \approx \mathcal{M}$.

Proof. Let f be a functional simulation from $\mathcal{T}$ to $\mathcal{S}$ and let g be a functional simulation from $\mathcal{T}$ to $\mathcal{M}$. Define relation $R \subseteq Q^{\mathcal{S}} \times Q^{\mathcal{M}}$ by

$$(s, q) \in R \Leftrightarrow \exists t \in B \cup F^0 \cup \cdots \cup F^{k-1} : f(t) = s \wedge g(t) = q.$$

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 $i \in I$. Let $s' = \delta^{\mathcal{S}}(s, i)$ and $q' = \delta^{\mathcal{M}}(q, i)$. We need to show that $\lambda^{\mathcal{S}}(s, i) = \lambda^{\mathcal{M}}(q, i)$ and $(s', q') \in R$. Since $(s, q) \in R$, there exists a $t \in B \cup F^0 \cup \cdots \cup F^{k-1}$ such that $f(t) = s$ and $g(t) = q$. Since $F^0, \ldots, F^k$ are all complete, $\delta^{\mathcal{T}}(t, i) \downarrow$. 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. Let $t' = \delta^{\mathcal{T}}(t, i)$. 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) $t' \in B \cup F^0 \cup \cdots \cup F^{k-1}$. In this case, since $f(t') = s'$ and $g(t') = q'$, $(s', q') \in R$ follows directly from the definition of R.

(b) $t' \in F^k$. We first show that basis B has the same number of states as $\mathcal{S}$. Since all states from basis B are identified, and since the apartness relation is irreflexive, all basis states are pairwise apart. Let q and q' be two distinct states in B. Since $q \mathrel{\#} q'$, we may conclude by Lemma 2.7 that $f(q) \not\approx f(q')$. Thus in particular $f(q) \neq f(q')$ and so f restricted to B is injective.

Let u be a state of $\mathcal{S}$. Since A is a state cover for $\mathcal{S}$, there exists a $\sigma \in A$ with $\delta^{\mathcal{S}}(q_0^{\mathcal{S}}, \sigma) = u$. Since basis B is complete, there exists a state $v \in B$ with $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, \sigma) = v$. Using that f is a functional simulation, we can show by induction on the length of σ that $f(v) = u$. Thus f restricted to B is surjective.

This implies that f restricted to B is a bijection between B and $Q^{\mathcal{S}}$, which proves B has the same number of states as $\mathcal{S}$.

With the same argument that we used to prove that f restricted to B is injective, we may also show that g restricted to B is injective. Thus $g(B)$ contains exactly $|Q^{\mathcal{S}}|$ states. Now either $g(B \cup F^0)$ contains the same number of states as $g(B)$, which implies that $\mathcal{M}$ contains $|Q^{\mathcal{S}}|$ states, or $g(B \cup F^0)$ contains at least one extra state. In the latter case we may continue: either $g(B \cup F^0 \cup F^1)$ contains the same number of states as $g(B \cup F^0)$, which implies that we have seen all states of $\mathcal{M}$, or $g(B \cup F^0 \cup F^1)$ contains at least one extra state. Etc. Since $\mathcal{M}$ has at most k more states than $\mathcal{S}$, we may conclude that $g(B \cup F^0 \cup \cdots \cup F^{k-1})$ contains all states of $\mathcal{M}$.

This means that $B \cup F^0 \cup \cdots \cup F^{k-1}$ contains a state t'' such that $g(t'') = q'$. We claim that t' and t'' have the same candidate set. The proof is by contradiction. Assume $C(t') \neq C(t'')$. We consider two cases:

i. $t'' \in B$. Then $C(t'') = \{t''\}$ and $t'' \notin C(t')$. Hence $t' \mathrel{\#} t''$. But now Lemma 2.7 gives $g(t') \neq g(t'')$. This is a contradiction, and therefore $C(t') = C(t'')$.

ii. $t'' \in F^0 \cup \cdots \cup F^{k-1}$. Let $C(t'') = \{r\}$. Then $r \mathrel{\#} t'$ and $\neg(r \mathrel{\#} t'')$. Therefore, by the co-transitivity assumption (1), $t' \mathrel{\#} t''$. But now Lemma 2.7 gives $g(t') \neq g(t'')$. This is a contradiction, and therefore $C(t') = C(t'')$.

We claim that for any state $u \in B \cup F^0 \cup \cdots \cup F^{k-1}$, $C(u) = \{r\}$ implies $f(u) = f(r)$. The proof of the claim is by contradiction. Assume $f(u) = v' \neq v = f(r)$. Since f restricted to B is a bijection, there exists an $r' \in B$ with $r \neq r'$ and $f(r') = v'$. Then $u \mathrel{\#} r'$. Let σ be a witness. By definition of apartness $\lambda^{\mathcal{T}}(u, \sigma) \neq \lambda^{\mathcal{T}}(r', \sigma)$. But since f is a functional simulation and $f(r') = v' = f(u')$, $\lambda^{\mathcal{T}}(u, \sigma) = \lambda^{\mathcal{S}}(v', \sigma) = \lambda^{\mathcal{T}}(r', \sigma)$. Contradiction and therefore the claim follows.

The above claim implies that $f(t'') = s'$. This in turn implies that $(s', q') \in R$, which completes the proof that R is bisimulation.

The theorem now follows by application of Lemma 3.4.

Remark 3.6. The assumption that A is a state cover for $\mathcal{S}$ implies that $\mathcal{S}$ is connected, that is, all states are reachable from the initial state. The assumptions that B is complete and all states in B are identified, in combination with Lemma 2.7, imply that $\mathcal{S}$ is minimal. Connectedness and minimality of specifications are common assumptions in conformance testing.

Example 3.7. A simple example of the application of Theorem 3.5, is provided by the observation tree from Figure 3 for the specification $\mathcal{S}$ from Figure 1. This observation tree corresponds to a test suite T of 4 tests that is generated by the W-method for $k = 0$, as described in the introduction. Note that the co-transitivity property vacuously holds when $k = 0$. All other conditions of the theorem are also met: both B and F^0 are complete and all states in B and F^0 are identified. Therefore, according to our theorem, test suite T is 0-complete. We may slightly optimize the test suite by removing state 14 from the observation tree (i.e., replacing test $bbaa$ by test bba), since all conditions of the theorem are still met for the reduced tree.

Example 3.8. Probably the most interesting condition in Theorem 3.5 is the co-transitivity requirement. The Mealy machines $\mathcal{S}$ and $\mathcal{M}$ of Figure 4 illustrate why we need it. Note that these machines are not equivalent: input sequence

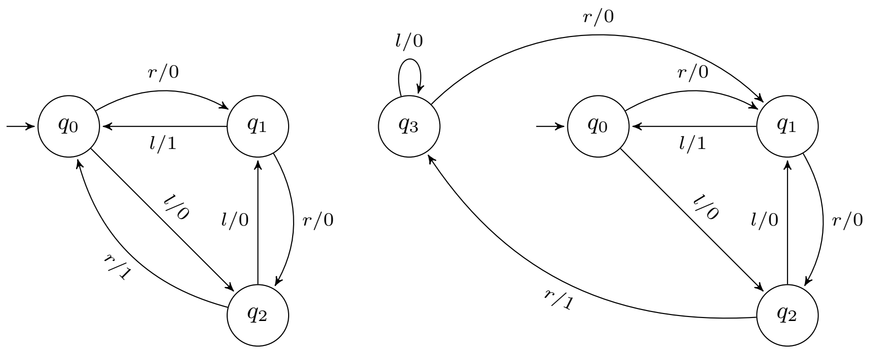

Fig. 4: A specification $\mathcal{S}$ (left) and a faulty implementation $\mathcal{M}$ (right).

$rrrlll$ distinguishes them. Mealy machine $\mathcal{M}$ has one more state than $\mathcal{S}$, so $k = 1$. The extra state q_3 of $\mathcal{M}$ behaves similar as state q_0 of $\mathcal{S}$, but is not equivalent. Figure 5 shows an observation tree $\mathcal{T}$ for both $\mathcal{S}$ and $\mathcal{M}$. 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, F^0 and F^1 are all complete, and all states in B, F^0 and F^1 are identified. However, the tree does not satisfy the co-transitivity condition as $t_{13} \# t_0$, but neither $t_0 \# t_6$ nor $t_{13} \# t_6$.

Example 3.9. In 2012, Arjan Blom, then a student at Radboud University, performed a security analysis of the E.dentifier2 system of the ABN AMRO bank, in

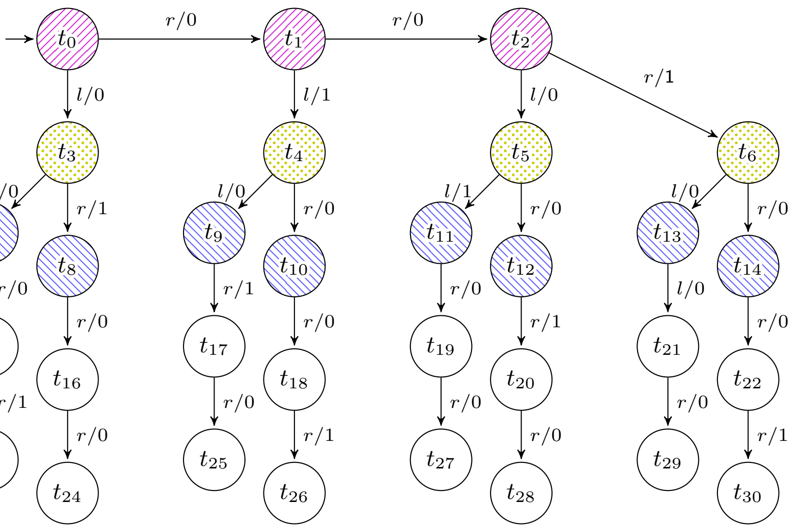

Fig. 5: Observation tree for FSMs $\mathcal{S}$ and $\mathcal{M}$ from Figure 4.

which customers use a USB-connected device – a smartcard reader with a display and numeric keyboard – to authorise transactions with their bank card and PIN code. He found a security vulnerability in the E.dentifier2 that was so serious that he even made it to the evening news on Dutch national TV. He did not use systematic testing techniques to find the vulnerability, but Chalupar et al. [1] used model learning to reverse engineer models of the E.dentifier2, demonstrating that this technique could have easily revealed the security problem.

Figure 6 shows the FSM S that specifies the required behavior of the E.dentifier2. There are three states $\{q_0, q_1, q_2\}$, five inputs $\{C, D, G, R, S\}$ and four outputs $\{C, L, T, OK\}$. We use commas to indicate multiple transitions. For instance, in S there is both a transition $q_1 \xrightarrow{C/OK} q_1$ and a transition $q_1 \xrightarrow{R/T} q_1$. We refer to Chalupar et al. [1] for a detailled explanation of the model. Note that $A = \{\epsilon, C, CD\}$ is an access sequence set for S, and sequence DR is a separating sequence for all pairs of states. Therefore, according to the W-method, the following test suite, which comprises 18 tests, is 0-complete:[1]

$$
\begin{aligned}
T = \{&DR, CDR, CDDR,\\
&CDR, DDR, GDR, RDR, SDR,\\
&CCDR, CDDR, CGDR, CRDR, CSDR,\\
&CDCDR, CDDDR, CDGDR, CDRDR, CDSDR\}
\end{aligned}
$$

[1] Test $CDGDR$ reveals the problem with the faulty implementation of the E.dentifier2 that was discovered by Arjan Blom.

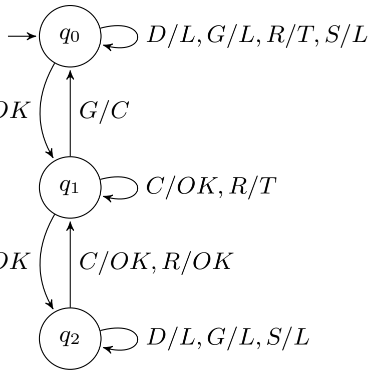

Fig. 6: FSM S that specifies the required behavior of the E.dentifier2.

Since we can safely omit redundant prefixes, we omit the three blue tests CDR, $CDDR$ and CDR. We claim that if we also omit the two green tests DR and $CDDDR$, the resulting test suite T' with 13 tests is still 0-complete. The argument is a bit subtle. Clearly, the three basis states are pairwise apart, using witnesses D and R. We know the response to distinguishing sequence DR for basis states t_1 ($T\ OK$) and t_2 ($L\ OK$), but not for initial state t_0 (as we decided to omit test DR). However, through test DDR, we know that in state 1 inputs DR trigger outputs LT. Under the assumption that $\mathcal{M}$ has three states, state 1 must be equivalent to either state t_0, t_1 or t_2. Therefore, the outcome of test DR in the initial state must be LT! Once we extend the observation tree with the inferred outcome of test DR, all F^0 states are easily identified. Since the basis and F^0 are complete, we may apply Theorem 3.5 to conclude that T' is 0-complete. Clearly, we cannot omit any other test, since then at least one F^0 state would no longer be visited, and $\mathcal{M}$ would no longer be uniquely determined.

Remark 3.10. Suppose observation tree $\mathcal{T}$ has N states. Then we can check in $\mathcal{O}(N^2)$ time for each pair of states of $\mathcal{T}$ whether they are apart or not. Using this information, we can check in $\mathcal{O}(N^2)$ time whether the conditions of Theorem 3.5 hold. This means that we can also check in $\mathcal{O}(N^2)$ time whether a test can be removed without compromising k-completeness. Note, however, that the size N of $\mathcal{T}$ grows exponentially in k.

4 Deriving Previous k-Completeness Results

The k-completeness of the W-method of Vasilevskii [20] and Chow [3] is a corollary of Theorem 3.5.

Corollary 4.1. *Let $\mathcal{S}$ be a minimal specification, let k be any natural number, let A be a state cover for $\mathcal{S}$, and let W be a characterization set for $\mathcal{S}$. Then the test suite $T = A \cdot I^{\leq k+1} \cdot W$ is k-complete for $\mathcal{S}$.*

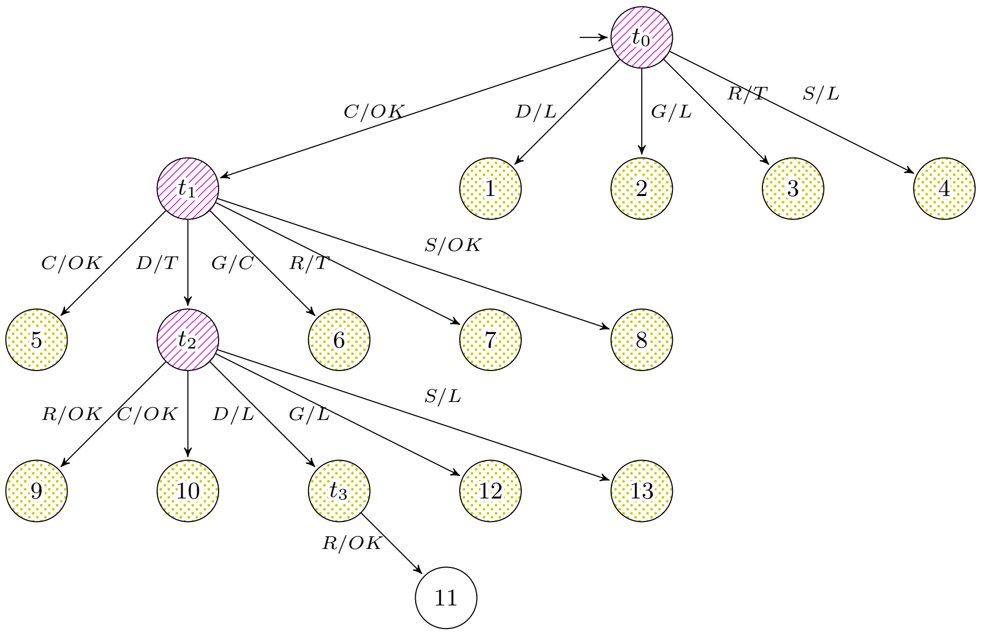

Fig. 7: Observation tree for test suite T' (for readability, we omitted the distinguishing sequence DR from the leaves in F^0)

Proof. Consider the observation tree $\mathcal{T}$ for $\mathcal{S}$ obtained by running all tests from T. Let $B, F^0, F^1, \ldots$ be the stratification of $Q^{\mathcal{T}}$ induced by A. Let ρ be an element from W. We check that the following assumptions of Theorem 3.5 hold:

1. Basis B is complete: Assume $\sigma \in A$. Then $\sigma\rho \in T$. Since $\mathcal{T}$ is constructed by running all tests from T, this implies that $\mathcal{T}$ contains a state q that is reached by input sequence σ. By definition, $q \in B$.
2. Frontier F^0 is complete: Suppose $q \in B$ and $i \in I$. Since q is a basis state, q is reached by a sequence $\sigma \in A$. Since $\sigma\ i\ \rho \in T$, $\delta^{\mathcal{T}}(q, i)\downarrow$.
3. For each $0 < j \leq k$, frontier F^j is complete: Suppose $q \in F^{j-1}$ and $i \in I$. By induction, we may show that there exists a basis state r and a sequence $\tau \in I^j$ such that q is reached from r via input sequence τ. Let σ be the access sequence for state r. Then $\sigma \cdot \tau \cdot i \cdot \rho \in T$. This implies $\delta^{\mathcal{T}}(q, i)\downarrow$, as required.
4. All states in B and $F^0, \ldots, F^k$ are identified: Suppose $q \in B \cup F^0 \cup \cdots \cup F^k$. Then by construction of T, for each $\rho \in W$, $\delta^{\mathcal{T}}(q, \rho)\downarrow$. Now suppose r and r' are two distinct states in B. Then, since W is a characterization set for $\mathcal{S}$, $\mathcal{T}$ is an observation tree for $\mathcal{S}$, A is a state cover for $\mathcal{S}$, and $B, F^0, F^1, \ldots$ is the stratification induced by A, there exists a $\rho \in W$ such that $\rho \vdash r \# r'$. Therefore, by the weak co-transitivity Lemma 2.8, either $q \# r$ or $q \# r'$. Since r and r' were chosen arbitrarily, this implies that state q is identified.

5. Co-transitivity: Suppose $r \in B$, $t' \in F^k$, $t'' \in F^0 \cup \cdots \cup F^{k-1}$ and $r \# t'$. By the previous item, state t' is identified, that is, there exists a r' such that $C(t') = r'$. Since $r \# t'$, we know that $r \neq r'$. Therefore, repeating the argument from the previous item, there exists a $\rho \in W$ such that $\rho \vdash r \# r'$. By construction of T, for each $\tau \in W$, $\delta^{\mathcal{T}}(t', \tau) \downarrow$ and $\delta^{\mathcal{T}}(t'', \tau) \downarrow$. Therefore, by weak co-transitivity and because $C(t') = r'$, $\rho \vdash r \# t'$., Another application of weak co-transitivity now gives $\rho \vdash r \# t''$ or $\rho \vdash t' \# t''$.

In order to prove that test suite T is k-complete, let $\mathcal{M}$ be a Mealy machine with at most k more states than $\mathcal{S}$. We show that $\mathcal{M}$ passes T if and only if $\mathcal{M} \approx \mathcal{S}$:

1. Assume $\mathcal{M} \approx \mathcal{S}$. Then $\mathcal{M}$ and $\mathcal{S}$ return the same result for each test. By definition, $\mathcal{M}$ passes a test σ if the resulting outputs are the same as for $\mathcal{S}$. Therefore, $\mathcal{M}$ passes all tests in T.
2. Assume $\mathcal{M}$ passes T. Then $\mathcal{T}$ is an observation tree for $\mathcal{M}$. Thus all conditions of Theorem 3.5 hold, and we may conclude that $\mathcal{M} \approx \mathcal{S}$.

Via similar arguments, k-completeness of the Wp-method of Fujiwara et al [5] and of the HSI-method of Luo et al [10] and Petrenko et al [14] also follows from our Theorem 3.5. The UIOv-method of Chan et al [2] is an instance of the Wp-method, and the ADS-method of Lee and Yannakakis [8] and the hybrid ADS method of Smeenk et al [16] are instances of the HSI-method. This means that, indirectly, k-completeness of these testing methods follows as well. The SPY-method of Simao, Petrenko and Yevtushenko [15] changes the prefixes in the HSO-method in order to minimize the size of a test suite, exploiting overlap in test sequences. Following [11], we believe the SPY-method should be considered as an optimization technique, orthogonal to the results of this article.

5 Conclusions and Future Work

We provided a sufficient condition for k-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-completeness of several methods for test suite generation that have been proposed in the literature.

Our characterization of k-completeness in terms of apartness triggers several questions. For instance:

1. Closest to our work is probably the result of Moerman [11, Proposition 2, Chapter 2], which provides sufficient conditions for a test suite of a certain shape to be k-complete. It would be interesting to see if this result of [11] follows from our result.
2. An intriguing question is whether it is possible to strengthen our result, that is, if weaker conditions exist that are still sufficient for k-completeness. An exciting perspective would be to come up with conditions that are not only sufficient but also necessary for k-completeness.

3. Our result suggest simple progress measures for performing a k-complete test suite, namely the sum of the number of elements of $B \cup F^0 \cup \cdots \cup F^k$ and the number of established apartness pairs required for state identification and co-transitivity. This progress measure in a way quantifies our uncertainty about the correctness of the implementation. A natural question then is to search for test queries that lead to a maximal increase of the progress measure. In order to reduce our uncertainty as fast as possible and/or to find bugs as quickly as possible, it makes sense to give priority to these tests.
4. 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. Of course, scalability may become an issue: for large specifications, large input alphabets and large values of k it may no longer be feasible to store the full observation tree in main memory. However, for many practical benchmarks (see e.g. [12]) it should not be a problem to handle the observation trees for k-complete test suites for $k = 2$ or $k = 3$.

Acknowledgements. Many thanks to Joshua Moerman, Jurriaan Rot and the anonymous reviewers for their feedback on an earlier version of this article.

References

1. Chalupar, G., Peherstorfer, S., Poll, E., de Ruiter, J.: Automated reverse engineering using Lego. In: Proceedings 8th USENIX Workshop on Offensive Technologies (WOOT'14), San Diego, California. IEEE Computer Society, Los Alamitos, CA, USA (Aug 2014)
2. Chan, W.Y.L., Vuong, C.T., Otp, M.R.: An improved protocol test generation procedure based on uios. p. 283–294. SIGCOMM '89, Association for Computing Machinery, New York, NY, USA (1989), https://doi.org/10.1145/75246.75274
3. Chow, T.: Testing software design modeled by finite-state machines. IEEE Trans. Software Eng. **4**(3), 178–187 (1978)
4. 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
5. Fujiwara, S., v. Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Trans. Software Eng. **17**(6), 591–603 (1991)
6. 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
7. Heyting, A.: Zur intuitionistischen Axiomatik der projektiven Geometrie. Mathematische Annalen **98**, 491–538 (1927)
8. Lee, D., Yannakakis, M.: Testing finite-state machines: State identification and verification. IEEE Trans. Comput. **43**(3), 306–320 (1994)
9. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines — a survey. Proceedings of the IEEE **84**(8), 1090–1123 (1996)

10. Luo, G., Petrenko, A., v. 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
11. Moerman, J.: Nominal Techniques and Black Box Testing for Automata Learning. Ph.D. thesis, Radboud University Nijmegen (Jul 2019)
12. Neider, D., Smetsers, R., Vaandrager, F.W., Kuppens, H.: Benchmarks for automata learning and conformance testing. In: Margaria, T., Graf, S., Larsen, K.G. (eds.) Models, Mindsets, Meta: The What, the How, and the Why Not? Lecture Notes in Computer Science, vol. 11200, pp. 390–416. Springer (2018)
13. 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)
14. 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)
15. da Silva 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
16. 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
17. Smetsers, R., Moerman, J., Jansen, D.N.: Minimal separating sequences for all pairs of states. In: Dediu, A., Janousek, J., Martín-Vide, C., Truthe, B. (eds.) Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Proceedings. Lecture Notes in Computer Science, vol. 9618, pp. 181–193. Springer (2016). https://doi.org/10.1007/978-3-319-30000-9_14
18. 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
19. 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
20. 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.)
21. Weyuker, E.J.: Assessing test data adequacy through program inference. ACM Trans. Program. Lang. Syst. **5**(4), 641–655 (1983), https://doi.org/10.1145/69575.357231