Using Automata Learning for Compliance Evaluation of Communcation Protocols on an NFC Handshake Example*

Stefan Marksteiner[1,2][0000−0001−8556−1541], Marjan Sirjani[2][0000−0001−5478−0987], and Mikael Sjödin[2][0000−0001−7586−0409]

[1] AVL List Gmbh, Graz, Austria stefan.marksteiner@avl.com
[2] Mälardalen University, Västerås, Sweden
{stefan.marksteiner,marjan.sirjani,mikael.sjodin}@mdu.se

Abstract. Near-Field Communication (NFC) is a widely adopted standard for embedded low-power devices in very close proximity. In order to ensure a correct system, it has to comply to the ISO/IEC 14443 standard. This paper concentrates on the low-level part of the protocol (ISO/IEC 14443-3) and presents a method and a practical implementation that complements traditional conformance testing. We infer a Mealy state machine of the system-under-test using active automata learning. This automaton is checked for bisimulation with a specification automaton modelled after the standard, which provides a strong verdict of conformance or non-conformance. As a by-product, we share some observations of the performance of different learning algorithms and calibrations in the specific setting of ISO/IEC 14443-3, which is the difficulty to learn models of system that a) consist of two very similar structures and b) very frequently give no answer (i.e. a timeout as an output).

Keywords: NFC · Automata Learning · Protocol Compliance · Bisimulation · Formal Methods

1 Introduction

In this paper we describe an approach of very thoroughly evaluating the compliance of Near-Field Communications (NFC)-based chip systems with the ISO/IEC 14443-3 NFC handshake protocol [10] using formal methods, concretely automata learning and equivalence checking. We present a tool chain that is easy

* This research received funding within the ECSEL Joint Undertaking (JU) under grant agreement No. 876038 (project InSecTT) and from the program "ICT of the Future" of the Austrian Research Promotion Agency (FFG) and the Austrian Ministry for Transport, Innovation and Technology under grant agreement No. 880852 (project LEARNTWINS). The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

to use - both the learning and the equivalence checking can run fully automatic. A complete automaton of the system-under-test (SUT) compared with a specification automaton modeled after the standard, provides a strong complement to conformance testing. The remainder of this paper structures as follows. First we provide its motivation and contribution. Section 2 gives an overview of basic concepts in this paper, including a formal definition of bisimulation for Mealy Machines as used in this paper. Section 3 describes the developed interface for automata learning of NFC systems, while Section 4 describes the learning setup including a comparison of different algorithms and calibrations to be most suitable for the specifics of the NFC handshake protocol. Section 5 shows real-world results, while Section 6 compare them to the works of others. Section 7, eventually, concludes the paper and gives and outlook on future work.

1.1 Motivation

As the NFC protocol is widely adopted in a broad variety of different, often security-critical, chip systems like banking cards, passports, access systems, etc., that use relatively weak hardware, a correct implementation is utterly important. While there are many works about security weaknesses in NFC (e.g., [14, 30]), also specifically regarding the ISO/IEC 14443-3 handshake (e.g., [8, 18]), there is few works on comprehensive testing (see Section 6). Assuring the correctness of the system is a principal step in the quest to trustworthy systems. As there is, to the best of our knowledge, no comprehensive works regarding assessment of the handshake protocols, as the fundament of secure protocols build atop, we aim for a strong verdict of ISO compliance for NFC systems. To make this verdict more scalable than manual modeling, yet strongly verified, we choose automata learning to automatically infer a formal model of the implementations under scrutiny. For the actual compliance checking, we use bisimulation and trace equivalence checks against a specification automaton from the ISO/IEC 14443-3 standard (a rationale is given in Section 2.2).

1.2 Contribution

Overall, this paper is on the interface between communications protocols, embedded systems and formal methods. This work provides the following contributions for people with scholarly or applied interest in this approach of strong compliance checking:

- Insights regarding the specifics of learning NFC using active automata learning
- An evaluation on the performance of different learning algorithms in systems with very similar structures
- Developing an NFC interface for a learning system
- An approach for automated compliance checking using bisimulation and trace equivalence

We saw the NFC handshake to be specific in two aspects: a) it consists of two parts that are very similar and hard to distinguish for Learners and b) the vast majority of outputs from a system-under-learning are timeouts. This has severe impact on the learning where we examined different algorithms and configurations. The maximum word length has an impact on correctly inferring an automaton: too short yields incomplete automata, too long seemed to have a negative performance impact. Surprisingly the L* algorithm [3] with Rivest/Schapire (LSR) closure [25] surpassed more modern ones in learning performance. For discovering deviations from the standard, the minimum word length was found to have an impact. Here, the TTT algorithm [12] performed best, also followed by LSR. We further created a concrete hardware/software interface using a Proxmark device and an abstraction layer for NFC systems. Lastly, we integrated bisimulation and trace equivalence checking into the learning tool chain, which enables completely automated compliance checking with counterexamples in the case of deviations from the standard.

2 Preliminaries

This section outlines the theoretical fundamentals of state machines and automata learning including a definition of equivalence and bisimilarity in the context of this paper. It further briefly describes the used framework and the basics and characteristics of the scrutinized protocol.

2.1 State Machines

A state machine (or automaton) is a fundamental concept in computer science. One of the most widely used flavors of state machines are Mealy machines, which describe a system as a set of states and functions of resulting state changes (transitions) and outputs for a given input in a certain state [20]. More formally, a Mealy machine can be defined as $M = (Q, \Sigma, \Omega, \delta, \lambda, q_0)$, with Q being the set of states, Σ the input alphabet, Ω the output alphabet (that may or may not identical to the input alphabet), δ the transition function ($\delta : Q \times \Sigma \to Q$), λ the output function ($\lambda : Q \times \Sigma \to \Omega$), and q_0 the initial state. The transition and output functions might be merged ($Q \times \Sigma \to Q \times \Omega$). An even simpler type of automaton is a deterministic finite acceptor (DFA) [19]. It lacks of an output (i.e. no Ω and no λ), but instead it has a set of accepted finishing states F, which are deemed as valid final states for an input word (i.e. sequence of input symbols), resulting in a definition of $D = (Q, \Sigma, \delta, q_0, F)$. The purpose is to define an automaton that is capable of deciding if an input word is a valid part of a language. A special subset of DFAs are combination lock automata (with the same properties) but the additional constraint that an invalid symbol in an input sequence would set the state machine immediately back into the initial state [22].

2.2 Transitions and Equivalence

An element of the combined transition/output function can be defined as 4-tuple ($\langle p, q, \sigma, \omega \rangle$) with $p \in Q$ as origin state of the transition, $q \in Q$ as destination state, $\sigma \in \Sigma$ as input symbol and $\omega \in \Omega$ as output symbol. Generally, to conform to a standard, a system must display the behavior defined in that standard. The ISO 14443-3 standard [10] describe the states of the NFC handshake with their respective expected input and result. . That means one can derive an automaton from this specification. The problem of determining NFC standard compliance can therefore be seen as comparing two (finite) automata. There is a spectrum of equivalences between Labeled Transition Systems (LTS) including automata. For being compliant with a standard, not necessarily every state and transition must be identical as long as the behavior of the system is the same. There might be learned automata that deviate from the standard automaton and still be compliant, e.g., if they are not minimal (the smallest automaton to implement a desired behavior). Figure 1 shows a very simple example of a three-state automaton and its behavior-equivalent (minimal) two-state counterpart.

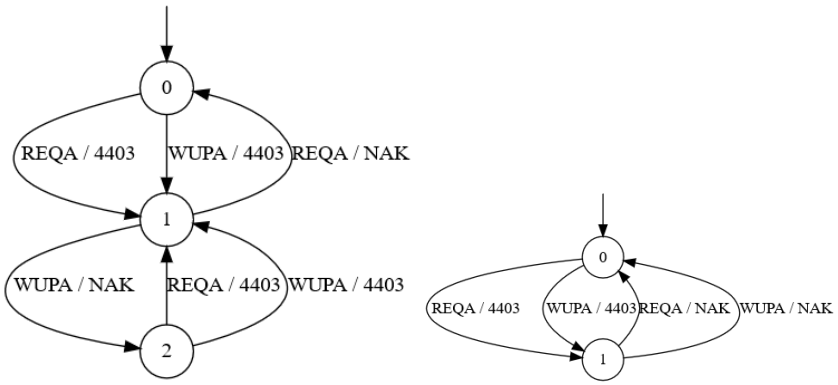

Fig. 1. Example for a partial automaton and its minimal counterpart.

To compare this type of equivalence between two LTS LTS_1 and LTS_2, commonly used are (various degrees of) simulation, bisimulation (noted as $LTS_1 \sim LTS_2$) and trace equivalence. Simulation means that one automaton can completely reproduce the behavior of the other, for the bisimulation, this relation becomes bidirectional (i.e. functional). Trace equivalence compares the respective output of automata. Just (uni-directional) simulation alone is not sufficient as this would only the presence *or* absence of a certain behavior with respect to the specification, while the standard compliance mandates both. Bisimilarity of two transition systems is originally defined for labeled transition systems (LTS), defined as $LTS = (S, Act, \rightarrow, I, AP, L)$, with S being the set of states, Act a set

of actions, $\rightarrow$ a transition function, I the set of initial states, AP a set of atomic propositions and L a labelling function.

Definition 1 (Bisimilarity). *Bisimlarity of two LTS LTS_1 LTS_2 is defined as exhibiting a binary relation $R \subseteq QxQ$, such that [4]:*

A) $\forall s_1 \in I_1 \exists s_2 \in I_2 \cdot (s_1, s_2) \in R$ *and* $\forall s_2 \in I_2 (\exists s_1 \in I_1 \cdot (s_1, s_2) \in R$.
B) for all $(s_1, s_2) \in R$ *must hold*
  1) $L_1(s_1) = L_2(s_2)$
  2) if $s_1\prime \in Post(s_1)$ *then there exists* $s_2\prime \in Post(s_2)$ *with* $(s_1\prime, s_2\prime) \in R$
  3) if $s_2\prime \in Post(s_2)$ *then there exists* $s_1\prime \in Post(s_1)$ *with* $(s_1\prime, s_2\prime) \in R$

Condition A of Definition 1 means that all initial states must be related, while Condition B means that for all related states the labels must be equal (1) and their successor states must be related (2-3). Formally the succession ($Post$) is defined as $Post(s, \alpha) = \{s\prime \in S | s \xrightarrow{\alpha} s\prime\}$ and $Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$, meaning the union of all action successions, which again are again the result the transition function with a defined action and state as input. As this is recursive, a relation of the initial states implies that all successor states are related. Since all reachable states are (direct or indirect) successor states of the initial states, this definition encompasses the complete LTS. We interpret Mealy machines as LTS using the output functions as labeling functions for transitions and the input symbols as actions, similar to [28]. Based on this, we define Mealy bisimilarity (M_1 M_2) for our purpose follows:

Definition 2 (Mealy Bisimilarity).

A) $q_{0_1} \in Q_1, q_{0_2} \in Q_2 \cdot (q_{0_1}, q_{0_2}) \in R$.
B) for all $q_1 \in Q_1, q_2 \in Q_2 \cdot (q_1, q_2) \in R$ *must hold*
  1) $\sigma \in \Sigma \cdot \lambda_1(q_1, \sigma) = \lambda_2(q_2, \sigma)$
  2) if $q_1\prime \in Post(q_1)$ *then there exists* $q_2\prime \in Post(q_2)$ *with* $(q_1\prime, q_2\prime) \in R$
  3) if $q_2\prime \in Post(q_2)$ *then there exists* $q_1\prime \in Post(q_1)$ *with* $(q_1\prime, q_2\prime) \in R$

As the transition function is dependent on the input, we define $Post(q, \sigma) = \delta(q, \sigma)$ and $Post(\sigma) = \bigcup_{\sigma \in \Sigma} Post(q, \sigma)$, which is essentially the same as for LTS brought into the notation of Section 2.1. There are a couple of different bisimulation types that differentiate by the handling of non-observable (internal) transitions (ordinarily labeled as τ transitions), e.g. strong and weak bisimulation, and branching bisimulation to give a few examples. This distinction is, however, theoretical in the context of this paper. The reason is that we intend to compare a specification, which consists of an automaton that does not contain any τ transitions, with an implementation that is externally (black box) learned, rendering τs unobservable. Therefore, two automata without any τs are compared directly, which makes this distinction not applicable. More precisely, from a device perspective, the type of bisimulation equivalence cannot be determined, as the SUTs are black boxes. This means that internal state changes (commonly denoted as τ) are not visible, which determines the kind of bisimulation. From a model perspective, the chosen comparison implies strong bisimulation (i.e the initial state

is related (formally, $q_{0_{M_l}} = q_{0_{M_s}}$) and all subsequent states are related as well (formally $Q = Q_{M_l} = Q_{M_s}; n = |Q|; \forall n \in Q | q_{n M_l} = q_{n_{M_s}}$).

Trace equivalence, on the other hand, means that two transitions systems produce the same traces for each same input.

Definition 3 (Trace equivalence). $Traces(LTS_1) = Traces(LTS_2)$

Although both bisimulation and trace equivalence might be principally capable of comparing a specification with an implementation automaton for determining the standard compliance, determining bisimulation is a problem to be solved in efficiently, whereas trace equivalence is PSPACE complete [2]. However, this might be negligible with a relatively low number of states and transitions. In any case, bisimulation implies trace equivalence ($LTS_1 \sim LTS_2$ implies $Traces(LTS_1) = Traces(LTS_2)$, but is finer than the latter [4]. For the purpose of this paper, we consider two automata equivalent if they are trace or bisimulation equivalent. In practice, we have obtained positive results with both bisimulation and trace equivalence (see Section 4.4). Therefore, trace equivalence is preferred as it is sufficient for standard compliance, but bisimilarity might be used in cases where more efficient checking algorithms are necessary.

2.3 Automata Learning

The classical method of actively learning automata of systems, was outlined in Angluin's pivotal work known as the L* algorithm [3]. This work uses a *minimally adequate Teacher* that has (theoretically) perfect knowledge of the SUT (in this case called System-under-learning – SUL) behind a *Teacher* and is allowed to answer to kinds of questions:

- *Membership queries* and
- *Equivalence queries.*

The membership queries are used to determine if a certain word is part of the accepted language of the automaton, or, in the case of Mealy machines, which output word will result of a specific input word. These words are noted in an observation table that will be made *closed* and *consistent.* The observation table consists of suffix-closed columns (E) and prefix-closed rows. The rows are intersected in short prefixes (S) and long prefixes ($S.\Sigma$). The short prefixes initially only contain the empty prefix (λ), while the long ones and the columns contain the members of the input alphabet. The table is filled with the respective outputs of prefixes concatenated with suffixes ($S.E$ or $S.\Sigma.E$). The table closed if for every long prefix row, there is a short prefix row with the same content ($\forall s.\sigma \in S.\Sigma \exists s \in S : s.\sigma = s$). The table is consistent if for any two equal short prefix rows, the long prefix rows beginning with these short prefixes are also equal ($\forall s, s\prime \in S \forall a \in \Sigma : s = s\prime \rightarrow s.a = s\prime.a$. A complete, closed and consistent table can be used to infer a state machine (set of states Q consists of all *distinct* short prefixes, the transition function is derived by following the suffixes). Even though this algorithm was initially defined for DFAs, it has

been adapted to other types of state machines (e.g., Mealy or Moore machines) [15]. Alternatively, some algorithms use a discrimination tree that uses inputs as intermediate nodes, states as leaf nodes, and outputs as branch labels, with a similar method of inferring an automaton. One of these algorithms, TTT[12], is deemed currently the most efficient [29]. Other widely used algorithms include a modified version of the original L* with a counterexample handling strategy by Rivest and Schapire [25], or the tree-based Direct Hypothesis Construction (DHC) [21] and Kearns-Vazirani (KV) [17] algorithms.

Once this is performed, the resulting automaton is presented to the Teacher, which is called equivalence query. The Teacher either acknowledges the correctness of the automaton or provides a counterexample. The latter is incorporated into the observation table or discrimination tree and the learning steps described above are repeated until the model is correct. To allow for learning black box systems, the equivalence queries in practice often consist of a sufficient set of conformance tests instead of a Teacher with perfect knowledge [24]. Originally for Deterministic Finite Automata, this learning method could be used to learn Mealy Machines [26]. This preferred for learning black box reactive systems (e.g. cyber-physical systems), as modeling these as Mealy is comparatively simple.

2.4 LearnLib

To utilize automata learning we use a widely adopted Java library called LearnLib [13]. This library provides a variety of learning algorithms (L* and variants thereof, KV, DHC and TTT), as well as various strategies for membership and equivalence testing (e.g., conformance testing like random words, random walk, etc.). The library provides Java classes for instantiating these algorithms and interfaces systems under test. The interface classes further allow for defining the input alphabets that the algorithm routines uses to factor queries used to fill an observation table or tree. Depending on the used algorithms, the library is capable of inferring DFAs, NFAs (Non-deterministic finite acceptors), Mealy machines or VPDAs (Visibly Pushdown Automata).

2.5 Near Field Communication

Near Field Communication (NFC) is a standard for simple wireless communication between close coupled devices with relatively low data rates (106, 212, and 424 kbit/s). One distinctive characteristic of this standard (operating at 13.56 Mhz center frequency) is that it, based on Radio-Frequency Identification (RFID), uses passive devices (proximity cards - PICCs) that receive power from an induction field from an active device (reader or proximity coupling device PCD) that also serves as field for data transmission. There are a couple of defined procedures that allow for operating proximity cards in presence of other wireless objects in order to exchange data [11]. One standard particularly defines two handshake procedures based on cascade-based anti-collision and card selection (called type A and type B), one of which NFC proximity cards must be compliant with [10]. This handshake is the particular target system-under-learning (SUL)

of this paper, with the purpose of providing very strong evidence for compliance. Due to the proliferation and the nature of the given system-under-learning, this paper concentrates on type A devices. Therefore, all statements on NFC and its handshake apply for type A only.

2.6 The NFC Handshake Automaton

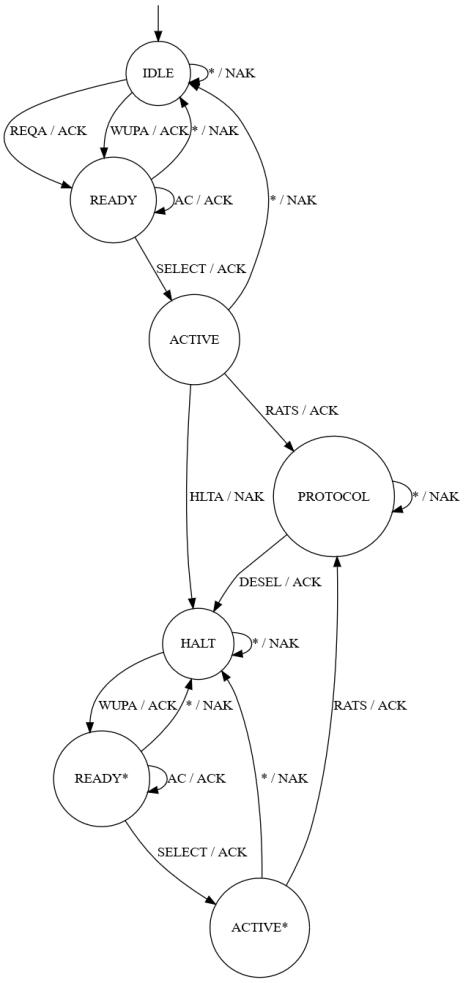

Fig. 2. NFC handshake automaton after ISO 14443-3 [10] augmented with abstract outputs. Note: star (*) as input means any symbol that is not explicitly stated in another outbound transition of the respective state.

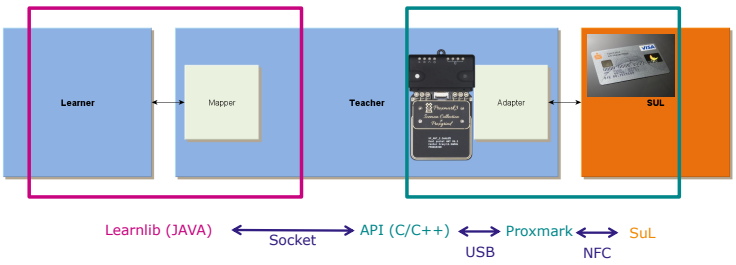

Fig. 3. NFC interface setup.

ISO 14443-3 contains a state diagram that outlines the Type A handshake procedure for an NFC connection (see Figure 2). This diagram is not a state machine of the types described in Section 2.1, for it lacks both output and final states. As we learn Mealy machines, we augmented it with abstract outputs (see Sections 4.2 and 4.4) to get a machine of the same type. The goal of the handshake is to reach a defined state in which a higher layer protocol (e.g. as defined in ISO 14443-4 [11]) can be executed (the *PROTOCOL* state). The intended way described in the standard to reach this state is: when coming into an induction field and powering up, the passive NFC device enters the *IDLE* state. After receiving a wake-up (*WUPA*) or request (*REQA*) message it enters the *READY* state. In this state, anti-collision (*AC*, remaining in that state) or card selection (*SELECT* going to the *ACTIVE* state) occur. In the latter state, the card waits for a request to answer-to-select (*RATS*), which brings it into said *PROTOCOL* state. In all of these states, an unexpected input would return the system to the *IDLE* state, no giving an answers (denoted as *NAK*). Based solely on ISO 14443-3 commands, the card should only leave this state after a *DESELECT* command, after which it enters the *HALT* state. Apart from a complete reset, it only leaves the *HALT* state after a wake-up (*WUPA*) signal (in contrast to the initial *IDLE* state, which also allows a *REQA* message). This brings it into the *READY** state, which again gets via a *SELECT* into the *ACTIVE** state that can be used to get to the *PROTOCOL* state again. The only difference between *READY* and *READY**, as well as *ACTIVE* and *ACTIVE** state is that it comes from the *HALT* instead of *IDLE* state. Similar to the first part of the automaton, an unexpected answer brings the state back to *HALT* without an answer (*NAK*).

Apart from the commands stated above that are expected by a card in the respective state, every other (i.e. unexpected) command would reset the handshake if its not complete (i.e. wrong commands from *IDLE, READY,* and *ACTIVE* states would lead back to the *IDLE* state, while *HALT, READY*,* and *ACTIVE** lead back to the *HALT* state and unexpected commands in the *PROTOCOL* state let it remain in that state. Even though this behavior of falling back into a base state resembles a combination-lock automaton or generally an accept-

ing automaton, we model the handshake as a Mealy Machine for the following reasons:

a) As we observe a black box, input/output relations are easier to observe than not intrinsically defined accepting states
b) The states are easier distinguishable: a variety of input symbols with the corresponding output may represent a broader signature than just if a state is accepting (apart from the transition to other states)
c) The output may processed at different level of abstraction (see Section 4.2)

There is also one specific feature to the NFC handshake protocol: unlike most communication protocols, an unexpected or wrong input yield to no output. This has an implication to learning, as a timeout will be interpreted as a general error message.

3 NFC Interface

As Learner, we use the algorithm implementations in the Learnlib Java library (see Section 2.4), configured as outlined in Section 4. To interact with the NFC SUL, a Proxmark RFID/NFC device (see Section 3.1) is used that works with an adapter written in C++ (see Section 3.2). Figure 3 provides an overview of the setup.

3.1 Learner Interface Device

The interface with an NFC SUL is established via Proxmark3. Proxmark3 is a pocket-size NFC device capable of acting as an NFC reader (PCD) or tag (PICC), as well as sniffing device [7]. Proxmark3 can be controlled from a PC, as well as, allowing firmware updates. Thus it allows us to construct the NFC frames needed for learning and establishing a connection to the learning library via a software adapter (see Section 3.2).

3.2 Adapter Class

The actual access to the NFC interface runs over a C++ program, running on a PC, based on a provided application that comes with the Proxmark device. As this application is open source, it was possible to modify it in order to adapt it for learning. The main interface to the Java-based Learner is a Socket connection that take symbols from the Learner (see Section 4.2) and concretizes them by translating the symbols into valid NFC frames utilizing functions from the *SendCommand* and *WaitForResponse* families. These functions send and receive, respectively, command data (i.e. concrete inputs, symbol for symbol) to the Proxmark device where the firmware translates it into frames and sends them to the SUL and proceeds vice versa for the response. This, however, turned out to create an error prone bottleneck at the connection between the PC application and the Proxmark device running over USB. Due to round-trip times

and timeouts, the learning was slowed down and occasional non-deterministic behavior was introduced, which jeopardized the learning process and made it necessary to repeat the latter (depending on the scrutinized system, multiple times, which hindered the overall learning greatly). Therefore, the Learner was re-implemented to send bulk inputs (i.e. send complete input words instead of single symbols), which improved the throughput significantly and solved non-determinism.

Firmware Modifications In order to be able to transfer traces word-wise instead of symbol-wise, significant modifications of the device's firmware were necessary. The standard interface of the device is designed for sending a single packet at one time (via a provided application on a PC) and delivering the answer back to the application via a USB interface. This introduces latency, which through the sheer amount of symbols sent in the learning process, has a significant performance impact. To reach the device's firmware with multiple symbols at once, we modulate the desired inputs into one sent message in Type-Length-Value (TLV) format (implemented types are with or without CRC and a specialized type for SELECT sequences) and modify the main routine of the running firmware to execute a custom function if a certain flag is set. This custom function deserializes the sent commands and sends them to the NFC SUT. Answers are modulated into an answer packet in length-value format, followed by subsequent answer messages containing precise logging and timestamps, if used. As NFC is a protocol that works with relatively low round-trip times and time outs these modifications, eliminating a great portion of the latency times of frequently used USB connections, boost the performance of the learning using different learning algorithms significantly (for a performance evaluation see Section 4.1).

4 Learning Setup

One distinctive attribute of ISO14443-3 with respect to learning is that it specifies to not give an answer on unexpected (i.e. not according to the standards specification) input. Ordinarily, the result of such a undefined input is to drop back to a defined (specifically the IDLE or HALT) state. In this sense, the NFC handshake resembles a combination lock. A positive output on the other hand, ordinarily consists of a standardized status code or information that is needed for the next phase of the handshake, e.g., parts of a card's unique identifier (UID). The non-answer to undefined is a characteristic feature of the NFC standard. This directly affects the learning because it yields many identical answers and efficient time-out handling is essential. It is therefore necessary to evaluate different state-of-the-art learning algorithms for their specific fitness (see Section 4.1) well as determining the optimal parameter set (Section 4.1). We scrutinize the main algorithms supported by Learnlib: classical L*, L* with Rivest/Schapire counterexample handling, DHC, KV and TTT - the latter two with linear search (L) and binary search (B) counterexample analysis.

Max. Word Length	Algorithm						
	L*-C	L*-RS	DHC	KV-L	KV-B	TTT-L	TTT-B
10	5.92	5.05	6.00	4.38	4.38	5.45	5.37
20	20.08	9.34	10.93	12.24	11.65	7.66	7.40
30	41.90	12.92	9.82	12.19	11.47	10.67	10.04
40	68.17	8.54	11.16	15.56	12.89	10.87	9.49
50	34.75	7.87	11.02	15.60	12.53	11.29	9.91
60	77.33	17.15	12.98	17.16	13.37	13.04	10.85
70	134.65	11.34	14.46	17.68	14.81	13.06	11.32

Table 1. Runtime (minutes) per algorithm and maximum word length.

4.1 Comparing Learning Algorithms and Calibrations

All of the algorithms can be parameterized regarding the membership and equivalence queries. The former are mainly defined via the minimum and maximum word length, while the equivalence queries (lack of a *perfect Teacher*), is determined by the method and number of conformance tests. Generally speaking, a too short (maximum) word length results in an incompletely learned (which, if the implementation is correct, should contain seven states). The maximum length, however, has a different impact on the performance for observation and tree-based algorithms: table-based are quicker with a short maximum word length, whereas for tree-based ones there seems to be a break-even point between many sent words and many sent symbols in our specific setting. Table 1 shows a comparison of the runtime of different algorithms with different maximum word lengths (in red the respective algorithm's shortest runtime that learned the correct 7-state model). Some of the non-steadiness in the results can be explained by the fact that some calibrations with shorter word lengths required more equivalence queries and, thus, refinement procedures. Table 2 shows the results with the best performing (correct) run of the respective algorithm. This, however, only covers the performance of learning a correct implementation. The opposite side, discovering a bug, shows a different picture. We therefore used a SUT with a slightly deviating behavior (see Section 5.3). This system is much more error-prone, needing significantly higher timeout values, resulting in higher overall runtimes. One key property in this case seems to be the minimum word length. Some of the algorithms by their require a lower minimum word length to discover than others. This has a significant impact with the special setting of getting relatively many timeouts, which is greatly aggravated by the necessary long timeout periods. With a minimum word length of 10 symbols, again the original L* with the Rivest/Schapire closing strategy was performing quickest, but discovered only 7 out of 10 states of the deviating implementation. DHC yielded a similar result. Both needed a word length of 20 to discover the actual non-compliant model, which was significantly less efficient in terms of runtime. The TTT and KV algorithms needed a minimum length of 10, however with quite some deviation in efficiency. While TTT was the best performing algorithm to learn the SUT's actual behavior model, KV was performing worst. The

Algorithm	L*-C (20)	L*-RS (10)	DHC (30)	KV-L (30)	KV-B (30)	TTT-L (30)	TTT-B (40)
States	7	7	7	7	7	7	7
Runtime (min)	20.08	5.05	9.82	12.19	11.47	10.67	9.49
Words	1137	282	539	496	451	468	382
Symbols	10192	2588	5124	7932	7607	6628	6213
EQs	2	3	2	5	5	4	4

Table 2. Performance evaluation of different algorithms for a compliant system with their respective fastest calibration in the given setting.

runtimes roughly correspond with the amount of sent symbols, in this case the a very long timeout has to be set to avoid non-determinism. The classical L* is not in the list, as the algorithm crashed after more than 24 hours of runtime. Table 3 provides an overview of minimum word lengths, run time, words, symbols and equivalence queries. Lower minimum word lengths yielded false negatives (i.e. the result showed a correct model with the deviation not uncovered).

4.2 Abstraction

Ordinarily, when applying automata learning to real-world systems, the input and output spaces are very large. To reduce the alphabets' cardinalities to a manageable amount, an abstraction function (∇), that transforms the concrete inputs (I) and outputs (O) to symbolic alphabets (Σ and Ω) using equivalence classes. Of all possible combinations of data to be send, we therefore concentrate on relevant input for the purpose of compliance verification. In the following we present some rationales for the chosen degree of abstraction through the input and output alphabets. These alphabets' symbols are abstracted and concretized via an according adapter class that translates symbols to data to be send (see section 3.2).

Input Alphabet For the input alphabet we use the one needed for successfully establishing a handshake (cf. Figure 2), according to the state diagram for Type-A cards in the ISO 14443-3 standard [10]:

- Wake-UP command Type A (WUPA)
- Request command, Type A (REQA)
- Anticollision (AC)
- Select command, Type A (SELECT)
- Halt command, Type A (HLTA)
- Request for answer to select (RATS)
- Deselect (DESEL)

The last two commands are actually defined in the ISO 14443-4 standard [11]. However, as the handshake's purpose is to enter and leave the protocol state, they are included in the 14443-3 state diagram and, consequentially, in our compliance verification.

Algorithm	L*-RS	DHC	KV-L	KV-B	TTT-L	TTT-B
Min Length	20	20	10	10	10	10
Runtime (min)	309.81	328.83	520.34	423.27	277.67	131.43
Words	575	855	952	679	688	616
Symbols	14637	15262	23867	19241	13353	11769
Eqs	5	3	6	6	5	5

Table 3. Performance evaluation of different algorithms for a non-compliant system with their respective fastest calibration in the given setting.

Output Alphabets In general, the output alphabet does not need to be defined beforehand. It simply consists of all output symbols observed by the Learner in a learning run. The Learner can derive the output alphabet implicitly. This means that if a system behaved non-deterministicly, the output alphabet could vary – although when learning Mealy machines, which are deterministic by definition, nondeterminism would jeopardize the Learner. The output alphabet has obviously to be defined (in the abstraction layer) when abstracting the output. Therefore, using raw output has the benefit of not having to define the alphabet beforehand. The raw method has one drawback: there are cards that use a random UID (specifically, this behavior was observed in passports). Every anti-collision (AC) and $SELECT$ yields a different output, which introduces non-deterministic behavior. This is not a problem with abstract output, as the concrete answer is abstracted away. We therefore tried a heavily abstracted output consisting of only two symbols, namely ACK for a (positive) answer and NAK for a timeout, which in this case means a negative answer (see Section 2.5). This solves the problem, but degrades the performance of the Learner, since states are harder to distinguish if the possible outputs are limited to two (aggravated by the similar behavior of certain states - see Section 2.6). This idea was therefore forfeit in favor of raw output for the learning. We still maintained this higher abstraction for the equivalence checking (see Section 4.4 for the reasoning). Raw output, however, retains this problematic non-determinism. We therefore introduce a caching strategy to cope with this issue. Whenever a valid (partial) UID is received as an answer to an anti-collision or select input symbol, we put it on one of two caches (one for partial UIDs from AC and one for full ones from $SELECT$ sequences). The Learner will subsequently only be confronted with the respective top entries of these caches. We therefore abstract away the randomness of the UID by replacing it with an actual but fixed one. This keeps the learning deterministic while saving the other learned UIDs for analysis, if needed.

4.3 Labeling and Simplification

An implementation that conforms to the standard will automatically labeled correctly, as the labelling function follows a standards-conform handshake trace:

a) label the initial state with $IDLE$,

b) from that point, find the state, where the transition with *REQA* as an input and a positive acknowledgement as an output ends and label it as *READY*,
c) from that point, find the endpoint of a positively acknowledged *SELECT* transition and label it as *ACTIVE*,
d) from that point, find the endpoint of a positively acknowledged *RATS* transition and label it as *PROTOCOL*,
e) from that point, find the endpoint of a positively acknowledged *DESELECT* transition and label it as *HALT*
f) from that point, find the endpoint of a positively acknowledged *WUPA* transition and label it as *READY**
g) from that point, find the endpoint of a positively acknowledged *SELECT* transition and label it as *ACTIVE**

If the labeling algorithm fails or there are additional states (which are out of the labeling algorithm's scope), this is an indicator for the learned implementation's non-compliance with the ISO 14443-3 standard (given that only the messages defined in that standard are used as an input alphabet - see Section 4.2).

To simplify the state diagram for better readability and analysis, we cluster the transitions of each states for output/target tuples and label the input for that mostly traveled tuple with a star ($*$). Normally that is the group of transitions that mark an unexpected input and transitions back to the IDLE or HALT state. This reduces the diagram significantly. Therefore, in those simplified diagrams, all inputs not marked explicitly in a state can be subsumed under the respective star ($*$) transition.

4.4 Compliance Evaluation

Proving or disproving compliance needs a verdict if a potential deviation from the standard violates the (weak) bisimulation relation. We use mCRL2 with the Aldebaran (.aut) format for bisimilarity and trace equivalence checking (as described in Section 2.2) [5]. As the Learnlib toolset provides to possibility to store the learned automata in a couple of formats, including Aldebaran, setting up the tool chain is easy, even though some re-engineering was necessary. Learnlib's standard function for exporting in the Aldebaran format does not include outputs. This accepts transitions as equal that are in fact not (as they distinguish only through the output). We therefore rewrote this function to use the transition's in the label of an LTS as well. mCRL2 comes with a model comparison tool that uses, amongst others, the algorithm of Jansen et al. [16] for bisimilarity checking. We therefore simply model the specification in form of the handshake diagram (see Figure 2) as an LTS with the corresponding Mealy's input and output as a label in the Aldebaran format and use the mCRL2 tool to compare it to automata of learnt implementations. The models of SUTs, although, could differ greatly event if the behavior is similar . Due to different UIDs the outputs to legit AC and SELECT commands would ordinarily differ between any two NFC cards. Also most other outputs might differ slightly. E.g., we observed some cards to respond to select with *4800*, others with *4400*. We therefore use

the higher abstraction level as described above and use only NAK and ACK as output, circumventing this problem. This way, inequalities as detected by the tool indicate non-compliance to the ISO 14443-3 standard of the scrutinized implementation. If a non-compliance (i.e. a missing or additional state or transition actually countering the bisimulation relation) is found, all we need is to do a simple conformance test. A trace of the non-compliant state/transition is trivial to extract from the automaton (see the example in Section 5.3). If that trace is executed on the system-under-test and actually behaves like predicted in the model, we have found the actual specification violation in the real system, disproving the compliance.

Alternatively, an actual positive verdict of compliance of a learned model is simple. A full compliance proof can be made when doing identity equivalence, that is comparing the learned model state by state and transition by transition with the model manually derived from the ISO 14443-3 standard. If every state and transition is equal, we consider the system as compliant. More formally, the learned machine M_l must be fully equal the specification machine M_s, i.e. $M_l = M_s \wedge (M_l = M_s \models Q_{M_l} = Q_{M_s} \wedge \Sigma_{M_l} = \Sigma_{M_s} \wedge \Omega_{M_l} = \Omega_{M_s} \wedge \delta_{M_l} = \delta_{M_s} \wedge \lambda_{M_l} = \lambda_{M_s} \wedge q_{0_{M_l}} = q_{0_{M_s}})$. This, obviously, is a simpler but stronger relation that is not coersive for ISO protocol compliance. The probability of learning (with a sufficient amount of conformance testing) an incorrect model that is still compliant with the standard is negligible.

5 Evaluation

In this section we briefly outline the achieved results with the described tool chain. We used serveral different NFC card systems for testing, which are described below. All of these systems have shown to be conform to the ISO14443-3 standard, except for the Tesla key fob.

5.1 Test Cards and Credit Cards

We used five different NFC test test cards by NXP (part of an experimental car access system) to develop and configure the Learner. Furthermore, we used two different banking cards, a Visa and a Mastercard debit. All of these cards are conform to the standard, with only minor differences. One of these deffenrences is replying with diffent ATQA to REQA/WUPA messages with 4400 and 4800 respectively. Overall, the results with these cards are very similar. Figure 4 shows an example of a learnt automaton (left side).

5.2 Passports

We also examined two different passports from European Union countries: one German and one Austrian. The main noticeable difference (at ISO 14443-3 levlel) between the other systems is that these systems answer to AC and SELECT inputs with randomly generated (parts of) UIDs. This implements a privacy

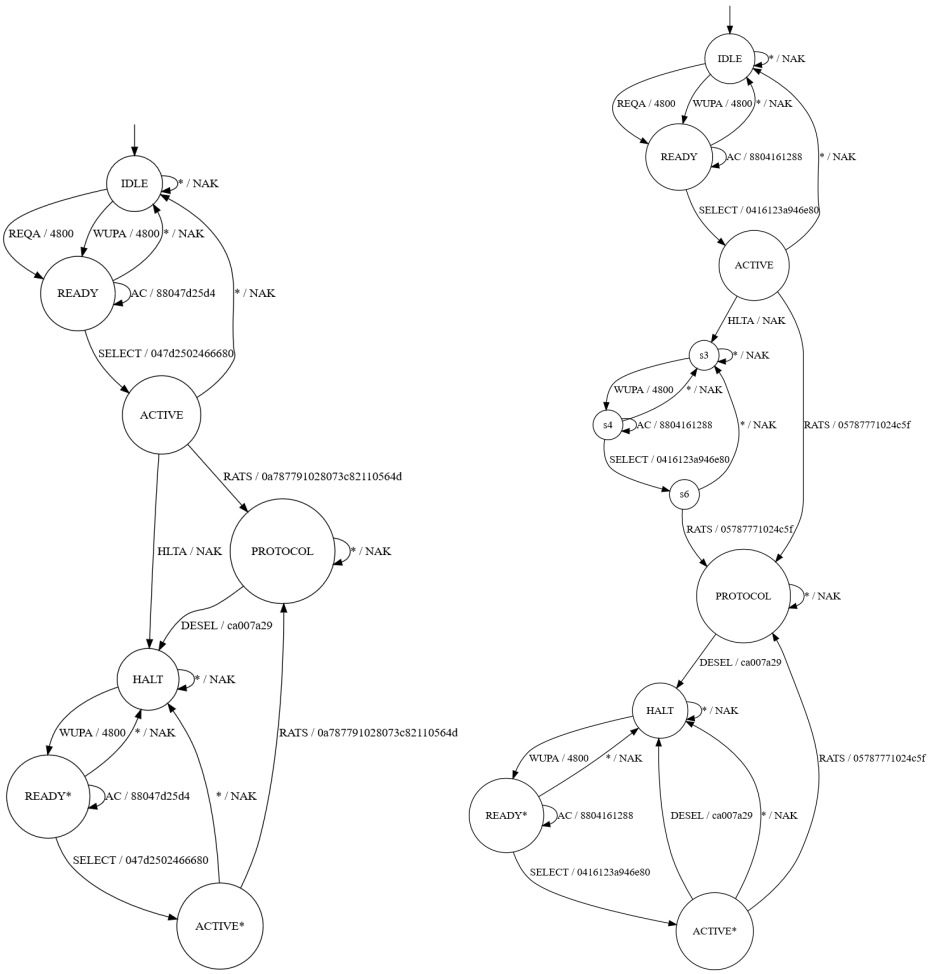

Fig. 4. Automaton of an NXP test card (left) and a Tesla car key fob (right) learnt with TTT.

feature to make passports less traceable. Without accessing the personal data stored on the device the passport should not be attributable. This, however, requires authentication.

5.3 Tesla Key Fob

Apart from significantly slower answers than the other devices, which required to adapt the timeouts to avoid nondeterministic behavior, the learned automaton slightly differs when learnt with the TTT algorithm. Figure 4 (right side) shows a model of a Tesla car key fob learnt with TTT. The (unnamed) states 3,4 and 6 are very similiar to the HALT, READY* and ACTIVE* states, respectively.

Apart from the entry points (HALTA from the ACTIVE state for the first and DESEL from the PROTOCOL state, respectively) these two structures are identical and in the reference model, those two transitions lead to the same state. However, the ACTIVE* transition allows for issuing a DESELECT command that actually returns a value (i.e. an ACK in the higher abstraction), which does not correspond to the standard.

The mCRL2 comparison tool rightfully identifies this model not to be bisimilar and trace equivalent with the specification. Using the according option, the tool also provided a counterexample in the form of the trace (⟨*REQA/ACK*⟩, ⟨*SELECT/ACK*⟩, ⟨*RATS/ACK*⟩, ⟨*DESEL/ACK*⟩, ⟨*WUPA/ACK*⟩, ⟨*SELECT/ACK*⟩, ⟨*DESEL/ACK*⟩). According to the specification, the last label should be ⟨DESEL/NAK⟩.

6 Related Work

There are other, partly theoretic, approaches of inferring a model using automata learning and comparing it with other automata using bisimulation algorithms. However, they target DFAs [6] or probabilistic transition systems (PTS) [9]. Neider et al. [23] contains some significant theoretic fundamentals of using automata learning and bisimulation for different types of state machines, including Mealys. It also contains the important observation that (generalized) Mealy Machines are bisimilar if their underlying LTS are bisimilar. Tappler et al. [28] used a similar approach of viewing Mealy Machines as LTS to compare automata regarding their bisimilarity. Similarly, bisimulation checking was also used to verify a model inferred from an embedded control software [27]. There is also previous work on using automata learning for inferring models of NFC cards [1], which concentrates on the upper layer (ISO/IEC 14443-4) protocol, dodging the specific challenges of the handshake protocol. Also there is no mentioning of automatic compliance checking in this approach. To the best of our knowledge, there is no comprehensive approach for compliance verification of the ISO/IEC 14443-3 protocol.

7 Conclusion

In this paper, we demonstrated the usage of automata learning to infer models of systems under test and evaluate their compliance with the ISO 14443-3 protocol by checking their bisimilarity with a specification. We described a learning interface setup, showed practical results and made interesting observations on the impact of the protocol specifics on learning algorithms' performances.

7.1 Discussion

Using our learning setup on real-world devices, we found little differences between the SUTs – all examined systems were compliant to ISO/IEC 14443-3.

Observed differences were mainly in the privacy-related random UIDs sent by passports and the slow answers and a slightly different automaton of the Tesla key fob. However, the scrutinized NFC handshake protocol has two characteristics that are distinct from other communications protocols: a) it does not send an answer on unexpected input and b) the automaton has two almost identical parts (IDLE/READY/ACTIVE and HALT/READY*/ACTIVE*) that pose challenges in learning. Supposedly these characteristics are responsible for the somewhat surprising finding that the L* algorithm with the Rivest/Schapire improvement surpasses more modern tree-based algorithms for correct systems. However, TTT performed best in finding a non-compliant system, which is the actual purpose of the testing and that the minimum word length has an impact on the ability to find incompliances. This might give some hints for optimization of learning strategies for similar structures.

7.2 Outlook

The compliance checking is but a first step towards assuring correctness and, subsequently, cybersecurity for NFC systems. Concretely, further research directions include test case generation using model checking and using the model to guide an intelligent fuzzer to leverage cybersecurity validation and verification (V&V). The target of these V&V activities are on the one hand upper layer protocols and on the other hand NFC reader devices to search for faults that might lead to exploitable security vulnerabilities. To talk to readers, because of the low latency of NFC communications, it is crucial to already know what to send before a conversation, which is satisfied by the predefined input words in the automata learning process.

References

1. Aarts, F., De Ruiter, J., Poll, E.: Formal Models of Bank Cards for Free. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops. pp. 461–468 (Mar 2013). https://doi.org/10.1109/ICSTW.2013.60
2. Aceto, L., Ingolfsdottir, A., Srba, J.: The algorithmics of bisimilarity. In: Advanced Topics in Bisimulation and Coinduction, pp. 100–172. Cambridge University Press (2011)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2), 87–106 (Nov 1987). https://doi.org/10.1016/0890-5401(87)90052-6
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (Apr 2008)
5. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 21–39. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1˙2

6. Chen, Y.F., Hong, C.D., Lin, A.W., Rümmer, P.: Learning to prove safety over parameterised concurrent systems. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 76–83 (Oct 2017). https://doi.org/10.23919/FMCAD.2017.8102244
7. Garcia, F.D., de Koning Gans, G., Verdult, R.: Tutorial: Proxmark, the swiss army knife for rfid security research: Tutorial at 8th workshop on rfid security and privacy (rfidsec 2012). Tech. rep., Radboud University Nijmegen, ICIS, Nijmegen (2012)
8. Hancke, G.: Practical attacks on proximity identification systems. In: 2006 IEEE Symposium on Security and Privacy (S&P'06). pp. 6 pp.–333 (May 2006). https://doi.org/10.1109/SP.2006.30
9. Hong, C.D., Lin, A.W., Majumdar, R., Rümmer, P.: Probabilistic Bisimulation for Parameterized Systems. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 455–474. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4˙27
10. International Organization for Standardization: Cards and security devices for personal identification – Contactless proximity objects – Part 3: Initialization and anticollision. ISO/IEC Standard "14443-3", International Organization for Standardization (2018)
11. International Organization for Standardization: Cards and security devices for personal identification – Contactless proximity objects – Part 4: Transmission protocol. ISO/IEC Standard "14443-4", International Organization for Standardization (2018)
12. Isberner, M., Howar, F., Steffen, B.: The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification. pp. 307–322. Lecture Notes in Computer Science, Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3"˙26
13. Isberner, M., Howar, F., Steffen, B.: The Open-Source LearnLib. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification. pp. 487–495. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4"˙32
14. Issovits, W., Hutter, M.: Weaknesses of the ISO/IEC 14443 protocol regarding relay attacks. In: 2011 IEEE International Conference on RFID-Technologies and Applications. pp. 335–342 (Sep 2011). https://doi.org/10.1109/RFID-TA.2011.6068658
15. Jacobs, B., Silva, A.: Automata Learning: A Categorical Perspective. In: van Breugel, F., Kashefi, E., Palamidessi, C., Rutten, J. (eds.) Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday, pp. 384–406. Lecture Notes in Computer Science, Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-06880-0˙20
16. Jansen, D.N., Groote, J.F., Keiren, J.J.A., Wijs, A.: An O(m log n) algorithm for branching bisimilarity on labelled transition systems. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 3–20. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-45237-7˙1
17. Kearns, M.J., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press (Aug 1994)
18. Maass, M., Müller, U., Schons, T., Wegemer, D., Schulz, M.: NFCGate: An NFC relay application for Android. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks. pp. 1–2. WiSec

'15, Association for Computing Machinery, New York, NY, USA (Jun 2015). https://doi.org/10.1145/2766498.2774984
19. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (Dec 1943). https://doi.org/10.1007/BF02478259
20. Mealy, G.H.: A method for synthesizing sequential circuits. The Bell System Technical Journal **34**(5), 1045–1079 (Sep 1955). https://doi.org/10.1002/j.1538-7305.1955.tb03788.x
21. Merten, M., Howar, F., Steffen, B., Margaria, T.: Automata Learning with On-the-Fly Direct Hypothesis Construction. In: Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation, vol. 336, pp. 248–260. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34781-8˙19
22. Moore, E.F.: Gedanken-Experiments on Sequential Machines. In: Automata Studies, AM-34, vol. 34, pp. 129–154. Princeton University Press (1956). https://doi.org/10.1515/9781400882618-006
23. Neider, D., Smetsers, R., Vaandrager, F., 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? Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday, pp. 390–416. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-22348-9˙23
24. Peled, D., Vardi, M.Y., Yannakakis, M.: Black Box Checking. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Formal Methods for Protocol Engineering and Distributed Systems: FORTE XII / PSTV XIX'99 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX) October 5–8, 1999, Beijing, China, pp. 225–240. IFIP Advances in Information and Communication Technology, Springer US, Boston, MA (1999). https://doi.org/10.1007/978-0-387-35578-8˙13
25. Rivest, R.L., Schapire, R.E.: Inference of Finite Automata Using Homing Sequences. Information and Computation **103**(2), 299–347 (Apr 1993). https://doi.org/10.1006/inco.1993.1021
26. Shahbaz, M., Groz, R.: Inferring Mealy Machines. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009: Formal Methods. pp. 207–222. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05089-3˙14
27. Smeenk, W., Moerman, J., Vaandrager, F., Jansen, D.N.: Applying automata learning to embedded control software. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) Formal Methods and Software Engineering. pp. 67–83. Springer International Publishing, Cham (2015)
28. Tappler, M., Aichernig, B.K., Bloem, R.: Model-Based Testing IoT Communication via Active Automata Learning. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). pp. 276–287 (Mar 2017). https://doi.org/10.1109/ICST.2017.32
29. Vaandrager, F.: Model learning. Communications of the ACM **60**(2), 86–95 (Jan 2017). https://doi.org/10.1145/2967606
30. Vila, J., Rodríguez, R.J.: Practical Experiences on NFC Relay Attacks with Android. In: Mangard, S., Schaumont, P. (eds.) Radio Frequency Identification. pp. 87–103. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015)