



# It's Not a Feature, It's a Bug: Fault-Tolerant Model Mining from Noisy Data

Felix Wallner
felix.wallner@ist.tugraz.at
Graz University of Technology
Graz, Austria

Bernhard K. Aichernig
aichernig@ist.tugraz.at
Graz University of Technology
Graz, Austria

Christian Burghard
christian.burghard@avl.com
AVL List GmbH
Graz, Austria

## ABSTRACT

The mining of models from data finds widespread use in industry. There exists a variety of model inference methods for perfectly deterministic behaviour, however, in practice, the provided data often contains noise due to faults such as message loss or environmental factors that many of the inference algorithms have problems dealing with. We present a novel model mining approach using Partial Max-SAT solving to infer the best possible automaton from a set of noisy execution traces. This approach enables us to ignore the minimal number of presumably faulty observations to allow the construction of a deterministic automaton. No pre-processing of the data is required. The method's performance as well as a number of considerations for practical use are evaluated, including three industrial use cases, for which we inferred the correct models.

## CCS CONCEPTS

• **Theory of computation → Logic and verification**; **Formal languages and automata theory**.

## KEYWORDS

Automata Learning, SAT solving, Partial Max-SAT, Model Inference, Non-Determinism

**ACM Reference Format:**
Felix Wallner, Bernhard K. Aichernig, and Christian Burghard. 2024. It's Not a Feature, It's a Bug: Fault-Tolerant Model Mining from Noisy Data. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3623346

## 1 INTRODUCTION

The problem of inferring Finite State Machines (FSMs) from a set of execution traces is well known: Originally proposed by Biermann and Feldman [9] as a constraint satisfaction problem (CSP) and then as a Boolean satisfiability problem (SAT) by Grinchtein et al. [17], which was further improved by others [5, 19], mining behavioural models from data is a well-researched topic [39]. Such models may be used for model-based development, verification and monitoring among other purposes in industry. In practice, however, the data that is used for behavioural model inference is often not perfectly deterministic. Faults such as message loss or other environmental influences, which may not be immediately apparent, may introduce noise into the data. This prevents many other deterministic inference methods, such as RPNI [41], from mining models from said data. Other approaches exist to learn stochastic [30] or non-deterministic automata [51], however, these methods encode the noise directly into the automata, which is often not desired, especially if it is known in advance that the underlying system is deterministic. In fact, one of the requirements from our industrial partner is to ignore or filter out the noise to learn the underlying FSM. At this point one has to either give up or perform preprocessing on the data, which may require domain knowledge.

In this paper, we present a novel approach to mine behavioural models from noisy data by ignoring the *least amount of steps necessary* in order to make it deterministic. The approach is such that no preprocessing regarding the noise is necessary. The assumptions about our underlying system are that it is deterministic and that faults appear sporadically and randomly, i.e., not systematically, with equal likelihood at any point in our data. The most typical noise with such properties is message loss, which we focus on during benchmarks by dropping input-output pairs from traces.

Additionally, it is possible to mine a model from a single long execution trace instead of multiple samples. While other approaches can work with single traces [18] they usually cannot deal with noise in the data. Two of our use cases demonstrate the use of our method on such single trace cases of industrial measurement devices.

Finally, our approach may also be used to extract a type of stochastic automaton, which includes the noise and the frequencies of the transitions used, as an addition to the fully deterministic models in order to better evaluate the types and amount of noise in the input data.

The research questions we want to answer are:

RQ 1: Is it possible to mine models from noisy data using Partial Max-SAT?
RQ 2: Which considerations have to be taken into account when applying the method in practice?
RQ 3: How performant is such an approach?

After presenting the preliminaries in Section 2, we answer these questions by means of the following contributions: (1) We present a novel approach to infer models from noisy data using Partial Max-SAT (Section 3). (2) We further present guidelines on how this method can be used in practice (Section 4). (3) We evaluate the method's performance (Section 5). The experiments show that despite the well-known complexity limitations of (Partial) Max-SAT, i.e., the problem being NP-hard [25], we were able to mine three relevant industrial models from measurement devices and communication protocols, as well as benchmark models of sizes



*ICSE '24, April 14–20, 2024, Lisbon, Portugal*

ACM ISBN 979-8-4007-0217-4/24/04.
https://doi.org/10.1145/3597503.3623346

up to 17 states and traces up to 6000 steps in length within several minutes. Longer traces or more states quickly lead to timeouts after one hour.

## 2 PARTIAL MAX-SAT FOR MOORE MACHINES

### 2.1 Moore Machines

Our algorithm assumes that the system we want to learn is deterministic and can be modelled as a Moore machine, as first described by Moore [33]. These are a type of FSM which can classically be defined as a 6-tuple, where the Moore machine $\mathcal{M}$ is:

$$\mathcal{M} = (I, O, S, s_0, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}})$$

where $I$, $O$ and $S$ are the finite, non-empty sets of input symbols, output symbols and states, respectively, $s_0 \in S$ is the initial state and $\delta_{\mathcal{M}} : S \times I \to S$ and $\lambda_{\mathcal{M}} : S \to O$ are the state transition and the output function, respectively. Importantly, the output function depends solely on the current state of the machine. A Moore machine is *input-complete* if $\delta_{\mathcal{M}}$ is fully defined.

We define a *trace* $t$ to be $t = \langle o_0, (i_1, o_1), (i_2, o_2), \ldots, (i_m, o_m)\rangle$ with $o_j \in O$ and $i_j \in I$. Note that the index of each trace starts with 0, as the first element of the trace is the initial output $o_0$, which is the output of the initial state $s_0$, followed by $m$ steps, i.e., input-output pairs, hence the length of the trace $|t| = m + 1$. For a given index $k \in [1, m]$ we will write $i_k$ and $o_k$ for the $k^{th}$ input and output of a trace, respectively.

Additionally, we define a *non-deterministic* Moore machine to use a transition relation $\delta_{\mathcal{M}} : S \times I \times S$ instead of a function. If the transitions of a non-deterministic Moore machine are also labelled with their occurrence frequencies or probabilities, we obtain a *stochastic* Moore machine. Note that the latter corresponds to a Markov-decision process.

### 2.2 Partial Max-SAT

The aim of traditional SAT solving is to find an assignment of variables for a given set of clauses in conjunctive normal form (CNF) such that the formula becomes true, i.e., satisfiable, or to find that such an assignment does not exists, i.e., the formula is unsatisfiable. Partial Max-SAT (PMSAT) is a mixture of the SAT and Max-SAT problems, insofar as a PMSAT problem consists of two sets of clauses: one which has to be satisfied using SAT and one for which the number of fulfilled clauses has to be *maximised* using Max-SAT. The clauses in the former set are called *hard clauses* while the ones in the latter are called *soft clauses*. Solving a PMSAT problem comes down to finding an assignment that satisfies the maximum number of soft clauses while also satisfying all hard clauses [4, 14].

*Quantifiers.* To more concisely describe the multitude of clauses necessary to build the PMSAT problem we use quantifiers. We resolve all quantifiers through enumeration such that $\forall x \in X : \mathcal{P}(x)$ corresponds to the CNF formula $\mathcal{P}(x_1) \wedge \mathcal{P}(x_2) \wedge \ldots \wedge \mathcal{P}(x_n)$, where each $\mathcal{P}(x_j)$ is a propositional variable and $\overline{\mathcal{P}(x_j)}$ its negation.

*Exactly One.* We define $\exists! x \in X : \mathcal{P}(x)$ to mean that we require exactly one of the $n$ variables $\{\mathcal{P}(x) : x \in X\}$ to be true, which will result in the following clauses: $(\mathcal{P}(x_1) \vee \mathcal{P}(x_2) \vee \ldots \vee \mathcal{P}(x_n))$ to require at least one variable to be true and $\forall i \in [1, n{-}1], \forall j \in [i{+}1, n] : (\overline{\mathcal{P}(x_i)} \vee \overline{\mathcal{P}(x_j)})$ to prevent any pair-wise combination of variables to be true at the same time. For example, $\exists! x \in \{x_1, x_2, x_3\} : \mathcal{P}(x)$ would result in $(\mathcal{P}(x_1) \vee \mathcal{P}(x_2) \vee \mathcal{P}(x_3)) \wedge (\overline{\mathcal{P}(x_1)} \vee \overline{\mathcal{P}(x_2)}) \wedge (\overline{\mathcal{P}(x_1)} \vee \overline{\mathcal{P}(x_3)}) \wedge (\overline{\mathcal{P}(x_2)} \vee \overline{\mathcal{P}(x_3)})$.

## 3 MODEL MINING WITH PARTIAL MAX-SAT

### 3.1 Variables

We define the following four classes of Boolean variables used by the solver in order to more clearly describe their meanings. Each instance of these terms corresponds to a single variable in the solver that can either be true or false:

- $\lambda(s, o)$ is true iff state $s$ has output $o$.
- $\delta(s, i, s')$ is true iff state $s$ transitions to state $s'$ on input $i$.
- $\omega(t, k, s)$ is true iff the post-state after $k$ steps in trace $t$ corresponds to $s$.
- $\mathcal{G}(t, k)$ is true iff step $k$ of trace $t$ is considered a *glitch*, i.e., does not conform to the deterministic transition function.

For readability purposes we use these terms directly in place of Boolean variables. For example, $\lambda(s', \text{acknowledge}) = \mathit{true}$ would mean that the state $s'$ has output 'acknowledge'. $\delta(s_x, \text{connect}, s_y) = \mathit{false}$ would mean that there exists no transition from state $s_x$ with input 'connect' to state $s_y$. A formula to describe that the output of a single state $s'$ should be either $a$ or $b$ would be: $(\lambda(s', a) \vee \lambda(s', b)) \wedge (\overline{\lambda(s', a)} \vee \overline{\lambda(s', b)})$, which would be equivalent to $\lambda(s', a) \oplus \lambda(s', b)$.

### 3.2 SAT Formalisation

Let $S = \{s_0, s_1, \ldots, s_{n-1}\}$ be a set of $n$ states and $T$ a set of traces over our inputs $I$ and outputs $O$. For the given sets $S, I, O$ and $T$, we may now build our SAT problem in the form of a CNF such that its assignment will correspond to an automaton with $n$ states that produces the given traces with the minimum number of glitches possible:

Each state of a valid automaton must have exactly one output:

$$\forall s \in S : \exists! o \in O : \lambda(s, o) \tag{1}$$

Every state must have a single deterministic transition on each state-input pair:

$$\forall s \in S : \forall i \in I : \exists! s' \in S : \delta(s, i, s') \tag{2}$$

and after each step in a trace the automaton must be in exactly one state:

$$\forall t \in T : \forall k \in [0, |t|) : \exists! s \in S : \omega(t, k, s) \tag{3}$$

Each output $o_k$ corresponds to the reached state $s$ of that given step in the trace:

$$\forall t \in T : \forall k \in [0, |t|) : \forall s \in S : (\omega(t, k, s) \implies \lambda(s, o_k)) \tag{4}$$

and for each step in the trace the predecessor state must have a valid transition to the successor state with the observed input. Otherwise, the step is considered a glitch:

$$\forall t \in T : \forall k \in [1, |t|) : \forall s, s' \in S : \quad (5)$$
$$(\omega(t, k-1, s) \wedge \omega(t, k, s') \implies \delta(s, i_k, s') \vee \mathcal{G}(t, k))$$

Finally, we want every step to not be a glitch if possible, however, we add this restriction as *soft clauses* in order to allow solutions even if we have too few states to explain the traces in full or in case the traces contain real non-determinism:

$$\forall t \in T : \forall k \in [1, |t|) : \textbf{soft} : \overline{\mathcal{G}(t, k)} \quad (6)$$

*Reliable Initial State.* The starting state of the given traces requires consideration. Usually, the initial state is the same over all traces and the initial output observed in all traces is also the same. This allows for an additional constraint: For each trace the first output before the first step at index 0 corresponds to the output of the initial state $s_0$:

$$\forall t \in T : \omega(t, 0, s_0) \quad (7)$$

However, if the initial state may differ between traces, i.e., we cannot rely on starting in the same state, then this constraint can be omitted, which will allow each trace to start at any state in the automaton.

*Optimisation.* Finally, an important optimisation that improves performance stems from the following consideration: If our set of outputs $O$ is built from the traces that contain our observations, then each output must appear at least in one state in the final automaton, otherwise we could not observe it. We may therefore assign each of the first $|O|$ states directly to one corresponding output:

$$\forall p \in [0, |O|) : \lambda(s_p, o_p) \quad (8)$$

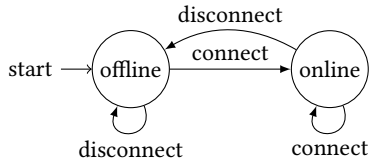
Note that, in contrast to Equation 3, $o_p$ refers to the $p$-th output in $O$. The indexing of $O$ is mostly arbitrary, however, if the constraint in Equation 7 is used, then the state $s_0$ must receive the corresponding initial output $o_0$ from the beginning of the traces. This optimisation strongly constrains the search space by preventing the assignment of different state labels to the same combinations of outputs. For example, instead of $\lambda(s_0, o_0) \wedge \lambda(s_1, o_1)$ or $\lambda(s_1, o_0) \wedge \lambda(s_0, o_1)$ only the first assignment is considered when using the constraint. This also makes assignment of outputs to states trivial in case of $|O|$ states, leaving only a single possible assignment. Increasing the number of states still keeps the first $|O|$ states fixed and allows for flexible output assignments for the additional states.

Table 1 summarises the clauses.

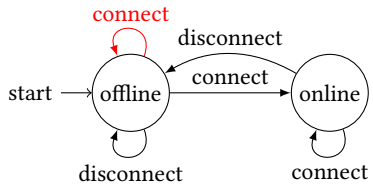
*Number of Clauses and Variables.* We define $t_{all} = \sum_{t \in T} |t|$ to be the total length of all traces and $t_{steps} = t_{all} - |T|$ to be the total number of steps in the traces, which do not include the initial state output of each trace. It can easily be seen from Equation 6 that the number of soft clauses is exactly $t_{steps}$. Because $\exists! x \in X$ is enumerated with $O(|X|^2)$ clauses, the total number of hard clauses can be given an upper bound of $O(|S|^3|I|)$ for Equation 2 and $O(t_{steps}|S|^2)$ for Equation 5 whereby the latter usually is the defining one due to $t_{steps} \gg |S|$. The number of unique variables, i.e., the sum of all unique $\lambda, \delta, \omega$ and $\mathcal{G}$, can be counted to be $|S| \times |O| + |S|^2 \times |I| + t_{all} \times |S| + t_{steps}$.

**Table 1: Summary of variables for inferring automata with states $S$ from traces $T$ with inputs $I$ and outputs $O$.**

| Eq. | Clauses | Enumeration |
|---|---|---|
| (1) | $\lambda(s, o)$ | $\forall s \in S : \exists! o \in O$ |
| (2) | $\delta(s, i, s')$ | $\forall s \in S : \forall i \in I : \exists! s' \in S$ |
| (3) | $\omega(t, k, s)$ | $\forall t \in T : \forall k \in [0, \lvert t\rvert) : \exists! s \in S$ |
| (4) | $\overline{\omega(t, k, s)} \vee \lambda(s, o_k))$ | $\forall t \in T : \forall k \in [0, \lvert t\rvert) : \forall s \in S$ |
| (5) | $\overline{\omega(t, k-1, s)} \vee \omega(t, k, s') \vee \delta(s, i_k, s') \vee \mathcal{G}(t, k)$ | $\forall t \in T : \forall k \in [1, \lvert t\rvert) : \forall s, s' \in S$ |
| (6) | $\textbf{soft} : \overline{\mathcal{G}(t, k)}$ | $\forall t \in T : \forall k \in [1, \lvert t\rvert)$ |
| (7) | $\omega(t, 0, s_0)$ | $\forall t \in T$ |
| (8) | $\lambda(s_p, o_p)$ | $\forall p \in [0, \lvert O\rvert)$ - $o_0$ initial output |




**(a) Moore machine of simple example server**




**(b) Inferred non-deterministic Moore machine of simple example server with glitch**

**Figure 1: Moore machines of simple example server.**

## 3.3 Example

By the following example, we want to demonstrate how a model can be inferred from traces:

Given the Moore machine in Figure 1a with $I = \{\text{connect}, \text{disconnect}\}$ and $O = \{\text{offline}, \text{online}\}$ let us assume we generated the following traces $T = \{t_0, t_1\}$ with:
$t_0 = \langle \text{offline}, (\text{disconnect}, \text{offline}), (\text{connect}, \text{online}) \rangle$ and
$t_1 = \langle \text{offline}, (\text{connect}, \text{online}), (\text{connect}, \text{online}), (\text{disconnect}, \text{offline}) \rangle$.

The given traces are representative, i.e., they contain enough information to fully determine the original automaton. Also, we must choose the number of states $n$ for the automaton we want to infer. For now, we choose the smallest possible $n$ in this case, namely $n = |O| = 2$ and build the following SAT problem for $S = \{s_0, s_1\}, I, O$ and $T = \{t_0, t_1\}$ using the definitions from Table 1:

(1) $(\lambda(s_0, \text{offline}) \vee \lambda(s_0, \text{online})) \wedge (\overline{\lambda(s_0, \text{offline})} \vee \overline{\lambda(s_0, \text{online})})$
$\wedge (\lambda(s_1, \text{offline}) \vee \lambda(s_1, \text{online})) \wedge (\overline{\lambda(s_1, \text{offline})} \vee \overline{\lambda(s_1, \text{online})})$
(2) $(\delta(s_0, \text{disconnect}, s_0) \vee \delta(s_0, \text{disconnect}, s_1)) \wedge$
$(\overline{\delta(s_0, \text{disconnect}, s_0)} \vee \overline{\delta(s_0, \text{disconnect}, s_1)}) \wedge$
$(\delta(s_0, \text{connect}, s_0) \vee \delta(s_0, \text{connect}, s_1)) \wedge \dots$
(3) $(\omega(t_0, 0, s_0) \vee \omega(t_0, 0, s_1)) \wedge (\overline{\omega(t_0, 0, s_0)} \vee \overline{\omega(t_0, 0, s_1)})$
$\wedge (\omega(t_0, 1, s_0) \vee \omega(t_0, 1, s_1)) \wedge (\overline{\omega(t_0, 1, s_0)} \vee \overline{\omega(t_0, 1, s_1)}) \wedge \dots$
(4) $(\overline{\omega(t_0, 0, s_0)} \vee \lambda(s_0, \text{offline})) \wedge (\overline{\omega(t_0, 0, s_1)} \vee \lambda(s_1, \text{offline})) \wedge$
$(\overline{\omega(t_0, 1, s_0)} \vee \lambda(s_0, \text{offline})) \wedge (\overline{\omega(t_0, 1, s_1)} \vee \lambda(s_1, \text{offline})) \wedge$
$(\overline{\omega(t_0, 2, s_0)} \vee \overline{\lambda(s_0, \text{online})}) \wedge (\overline{\omega(t_0, 2, s_1)} \vee \lambda(s_1, \text{online})) \dots$
(5) $(\overline{\omega(t_0, 0, s_0)} \vee \overline{\omega(t_0, 0, s_1)} \vee \delta(s_0, \text{disconnect}, s_1) \vee \mathcal{G}(t_0, 1)) \wedge$
$(\overline{\omega(t_0, 0, s_1)} \vee \overline{\omega(t_0, 0, s_2)} \vee \overline{\delta(s_1, \text{connect}, s_2)} \vee \overline{\mathcal{G}(t_0, 2)}) \wedge \dots$
(6) **soft:** $\mathcal{G}(t_0, 1) \wedge \mathcal{G}(t_0, 2) \wedge \overline{\mathcal{G}(t_1, 1)} \wedge \overline{\mathcal{G}(t_1, 2)} \wedge \mathcal{G}(t_1, 3)$
(7) $\omega(t_0, 0, s_0) \wedge \omega(t_1, 0, s_0)$
(8) $\lambda(s_0, \text{offline}) \wedge \lambda(s_1, \text{online})$

Giving the above SAT problem to a PMSAT solver yields the following solution, which contains all the variables assigned true by the solver: $\lambda(s_0, \text{offline}) \wedge \delta(s_0, \text{connect}, s_1) \wedge \delta(s_0, \text{disconnect}, s_0) \wedge \lambda(s_1, \text{online}) \wedge \delta(s_1, \text{connect}, s_1) \wedge \delta(s_1, \text{disconnect}, s_0) \wedge \omega(t_0, 0, s_0) \wedge \omega(t_0, 1, s_0) \wedge \omega(t_0, 2, s_1) \wedge \omega(t_1, 0, s_0) \wedge \omega(t_1, 1, s_1) \wedge \omega(t_1, 2, s_1) \wedge \omega(t_1, 3, s_0)$.

Interpreting this solution as an automaton, by looking at the assigned $\lambda$ state outputs and the $\delta$ transitions, yields the automaton in Figure 1a, which we correctly infer with zero glitches.

*Example with Glitches.* Now, let us assume we observe a third trace $t_2 = \langle \text{offline}, (\text{disconnect}, \text{offline}), (\text{connect}, \text{offline}) \rangle$.

This trace contains a step from state 'offline' with input 'connect' to state 'offline' which is obviously not part of the real automaton in Figure 1a. This incorrect observation may have happened due to any number of reasons: For example, the successful connection to the server in between the two steps was not logged due to message loss and as such made the trace non-deterministic.

At this point, regular SAT inference would report that the given problem with traces $T = \{t_0, t_1, t_2\}$ is unsatisfiable as no deterministic automaton exists. It is true that no such automaton exists, however, it would be more useful to receive the *best possible* deterministic automaton using *as much of the traces as possible* instead of not getting any result at all.

Building the SAT problem again with two states and traces $T = \{t_0, t_1, t_2\}$ analogous to before and solving it with PMSAT gives us the following solution:
$\lambda(s_0, \text{offline}) \wedge \delta(s_0, \text{connect}, s_1) \wedge \delta(s_0, \text{disconnect}, s_0) \wedge \lambda(s_1, \text{online}) \wedge \delta(s_1, \text{connect}, s_1) \wedge \delta(s_1, \text{disconnect}, s_0) \wedge \omega(t_0, 0, s_0) \wedge \omega(t_0, 1, s_0) \wedge \omega(t_0, 2, s_1) \wedge \omega(t_1, 0, s_0) \wedge \omega(t_1, 1, s_1) \wedge \omega(t_1, 2, s_1) \wedge \omega(t_1, 3, s_0) \wedge \omega(t_2, 0, s_0) \wedge \omega(t_2, 1, s_0) \wedge \omega(t_2, 2, s_0) \wedge \mathcal{G}(t_2, 2)$.

As can be seen from the solution, the second step in the trace $t_2$ was determined to be a glitch, for which no transition exists in the inferred automaton. Visualising this solution results in Figure 1a if only the $\delta$ variables are taken into account. We call transitions defined by the $\delta$ variables assigned true by a solver *dominant* transitions. An automaton using only dominant transitions is called *dominant automaton* moving forward.

We may also visualise the glitch $\mathcal{G}(t_2, 2)$ by adding the transition $\delta_g(s_0, \text{connect}, s_0)$, as seen in Figure 1b, by taking $i_2$ of trace $t_2$ and adding it between the states of steps $\omega(t_2, 1, s_0)$ and $\omega(t_2, 2, s_0)$. We will denote transitions traversed in the trace but marked as a glitch with $\delta_g$ and call them *glitched transitions*, which are drawn in red.

At this point the glitch in the trace and the automata in Figures 1a and 1b may be examined to determine the validity of the result or to evaluate the steps marked as glitches in the trace. In this example, the glitch stems from the incompatibility between traces $t_0$ and $t_2$, which cannot be resolved by adding more states to the inferred automaton, i.e., non-determinism instead of overgeneralisation. As we will show in the next section, some types of glitches may and should be resolved by inferring automata with more states.

In this section, we answered RQ 1, showing that it is possible to mine models from noisy data using the PMSAT encoding described above and demonstrating its use on an example.

## 4 PRACTICAL CONSIDERATIONS

The procedure above can be used to mine models from traces that include non-deterministic behaviour or noise given the number of states $n$ the inferred automaton should have. However, in practice the real number of states of the underlying system is not known in advance and therefore cannot be used as a parameter in the learning procedure. In this section we provide some guidelines on how to determine $n$ and on how to choose an automaton out of a range of automata with different $n$ that should be "as deterministic as possible", having few glitches and yet generalising the behaviour recorded in the traces as well as possible. We also provide general remarks about using the PMSAT inference procedure in practice and present an example on how to apply the guidelines.

### 4.1 Considerations for Choosing $n$

First, it is important to note that the smallest possible automaton that can be inferred from a trace has exactly $n = |O|$ states. Assuming that $O$ is calculated from the outputs observed in the traces, such that $O = \bigcup_{t \in T} \{o_k | k \in [0, |t|)\}$, and due to the fact that every state assigned to a step in the trace must have the corresponding output (Equation 4), then any call to the solver with $n < |O|$ will be unsatisfiable. Additionally, defining $I$ or $O$ to be different from the inputs and outputs observed in the trace respectively is bound to provide unsatisfiable or at least unhelpful results, as the trace then cannot be correctly represented.

Conversely, the solver will always produce a valid result for any $n \geq |O|$. Usually, the solver will use new states to encode glitches into dominant transitions and thereby decrease the number of glitches in the solution, which is of course the objective of the PMSAT algorithm. Alternatively, as there is no notion of reachability in our SAT encoding, new states may always be added by connecting the transitions of a new state to any other state in the automaton, thus creating an *unreachable* state, which is demonstrated in the example in Section 4.2. Therefore, inferred automata with increasing number of states $n$ will always have the same or decreasing number of glitches.

To better analyse and compare the automata with different $n$ we can use the solutions of the solver to build a *stochastic* Moore machine that not only includes glitches but also shows the *frequencies* of transitions, i.e., how often each transition was taken in the traces. This can be done by replaying the trace using the $\omega$ of the solution variables and counting the number of times a transition

was taken. Figures 2a, 2b and 2c show such stochastic Moore machines all learned from the same trace but with different $n$, where dominant transitions are black, glitched transitions are red and the frequencies of both are written in parentheses.

Ideally, we would get a **small** automaton, that has **few glitches** with **low transition frequencies** and for which dominant transitions have **high frequencies**. Small automata are preferable because they generalise better than larger ones. The larger the automaton the more observed behaviour will be encoded into states up to and including outliers, special cases or noise, which is often unwanted. Additionally, observing certain interactions more often in the traces gives us more confidence that said transition is real behaviour of the system and not noise. Thus, it is preferable for dominant transitions to have high frequencies. In contrast, if a transition that was marked as a glitch has a high frequency then this would imply said transition should be encoded into the automaton as a dominant transition.

The following guidelines (GL) inform about the best possible choice of $n$:

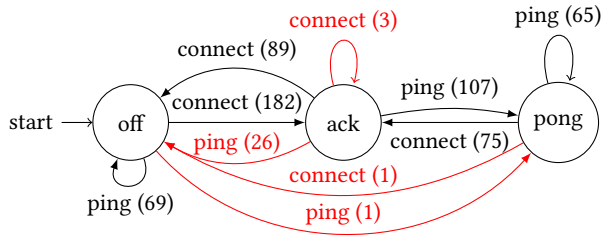
GL 1: Choose $n = |O|$ to infer the smallest possible automaton.
GL 2: If the solution has too many glitches, increment $n$.
GL 3: If the resulting automaton has frequently used glitches, increment $n$ (we have generalised too much).
GL 4: If the resulting automaton has too many low frequency dominant transitions, decrement $n$ (we have encoded outliers into the automaton that are likely to be glitches).

It is recommended to evaluate the *change* in glitches and frequencies instead of absolute numbers as, depending on the data, some automata might already have very high or low frequencies. For example, if the traces are not representative or complete, then even the smallest automaton might not be input-complete already and, as such, generally have lower dominant frequencies.

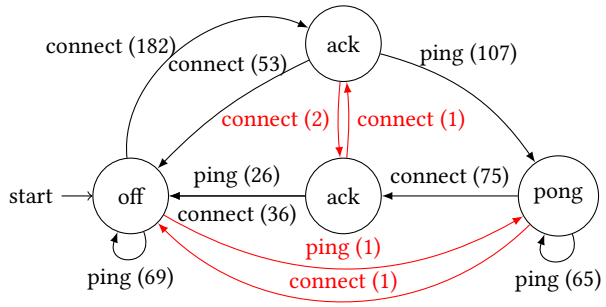
Just minimising the number of glitches, even if an automaton with zero glitches can be found, is often not recommended because such an automaton is often simply a tree of all given traces. This is especially true if an automaton should be inferred from only a single trace, in which case there does always exists an automaton with zero glitches in the form of a line. There exists a certain balance between *generalising* behaviour and *precisely* representing the given data that must be determined for each set of traces and each use case. This "best" result depends on the acceptable percentage of glitches in the traces and the desired frequencies. For example, if the desired result should be input-complete then all dominant transitions should have a frequency of at least one.

As already stated the inferred automata might include states that are not reachable via dominant transitions. If the number of such dominant reachable states $n_{reach} < n$ then this means that only glitches lead to the unreachable states. Most of the time the resulting automaton is not the best one, as the number of glitches is only artificially lowered without improving the dominant automaton. However, it is possible that some outputs are only observed once or very rarely, in which case the transitions into the states with these outputs could correctly be marked as glitches.

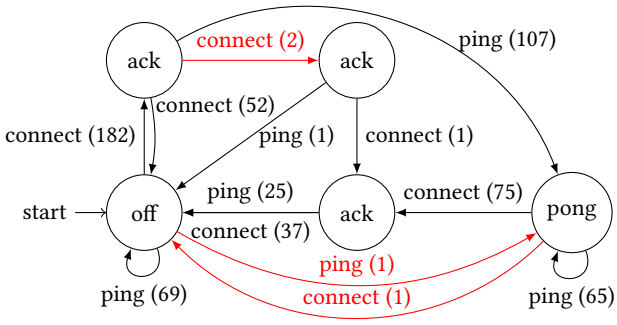
Lastly, automata with different $n$ may be checked for *bisimilarity*. We use the standard definition of bisimilarity [13], namely that two automata are bisimilar if both simulate each other. We only compare




**(a) Inferred stochastic Moore machine with $n = 3$ states of ping-pong server with 31 glitches**




**(b) Inferred stochastic Moore machine with $n = 4$ states of ping-pong server with 5 glitches**




**(c) Inferred stochastic Moore machine with $n = 5$ states of ping-pong server with 4 glitches**

**Figure 2: Stochastic Moore machines of ping-pong server.**

dominant automata for bisimilarity. In case two automata with different $n$ are bisimilar, the smaller automaton may be preferable.

## 4.2 Inferring Examples with Different $n$

A simple example ping-pong server serves as a demonstration of the use of the guidelines and transition frequencies above. Figures 2a, 2b and 2c are the automata inferred from a set of traces using an $n$ of three, four and five states, respectively. We chose to visualise the results as stochastic Moore machines in which the glitches are red and the frequencies are written in parentheses after the input of each transition. The sum of all steps over the set of traces used to

**Table 2: Statistics of inferring ping-pong server with different $n$ as parameter, with $n_{reach}$ dominant reachable states, number of glitches and different statistics of frequencies (fr.) for glitched $\delta_g$ and dominant $\delta$ transitions.**

| $n$ | $n_{reach}$ | # Glitches | Mean $\delta_g$ fr. | Max $\delta_g$ fr. | Min $\delta$ fr. |
|---|---|---|---|---|---|
| 3 | 3 | 31 | 7.75 | 26 | 65 |
| 4 | 4 | 5 | 1.25 | 2 | 26 |
| 5 | 4 | 4 | 1.33 | 2 | 1 |
| 6 | 4 | 4 | 1 | 1 | 0 |
| 7 | 4 | 4 | 1 | 1 | 0 |

infer these examples was 619. Thus, the sum of all frequencies is also 619 in each automaton. Five of these steps were faults, which is a little less than 1% of the total amount of steps.

Using the guidelines above, we first infer the smallest possible automaton with $n = |O| = 3$ in Figure 2a (GL 1). The solution for $n = 3$ has 31 glitches, which is equivalent to the sum of the frequencies on all glitched (red) transitions in the automaton, about 5% of all steps in our set of traces. 5% is already a high amount of glitches, which means we would tend towards increasing $n$ (GL 2), although it could be acceptable depending on the quality of the data and the specific use case. Additionally, the glitched 'ping' transition from 'ack' to 'off' has a very high frequency, which also prompts us to increase $n$ (GL 3).

Taking a look at Figure 2b we can see that this is a much better automaton in accordance with our guidelines: The number of glitches fell drastically from 31 down to five, approximately 1% of $t_{steps}$, and there are no longer any glitches with high frequencies, the highest being two in comparison to 26 before. Additionally, all the dominant transitions still have high frequencies meaning no outliers were encoded into states. This could be a good candidate for a final result.

Finally, we check the next automaton in Figure 2c with $n = 5$ states. Here the solver was able to reduce the number of glitches down to four. However, there are multiple indicators that this is not a good result: Firstly, the number of states actually reachable via dominant transitions $n_{reach}$ is only four instead of five. One 'ack' state can only be reached via the glitched 'connect' transition. This is a trick the solver uses quite frequently to reduce the number of glitches only if no other more meaningful improvement is possible. The 'connect' transition from the unreachable state to the other 'ack' state is now dominant and, as such, does not count towards the number of glitches anymore. Secondly, the two new dominant transitions only have a frequency of one, which implies that outliers or special cases were encoded into states (GL 4). Lastly, the dominant automata with $n = 4$ and $n = 5$ are bisimilar and as such are not distinguishable by inputs and outputs.

Taking a look at Table 2 to verify our analysis from above we can see the drastic changes in the number of glitches as well as in mean and maximum frequencies of $\delta_g$ between $n = 3$ and $n = 4$. Conversely, only a single glitch is removed between $n = 4$ and $n = 5$ while we suddenly have dominant transitions with a frequency of only one, which is the encoding of the glitch into a dominant transition.

Therefore, choosing between these automata, we would choose Figure 2b with $n = 4$ states. In this case, this would have been correct as the dominant automaton of this figure was the ground truth from which we generated a set of traces and randomly discarded input-output pairs to simulate faults in the data.

One last important point is that the dominant automata for $n = 4$, $n = 5$, $n = 6$ and $n = 7$ are all *bisimilar* and, as such, cannot be distinguished by observations alone. All four automata have only four reachable states. In a sense, the only valid choices are between the $n = 3$ and $n = 4$ automata as the others are bisimilar to $n = 4$.

That does of course not mean that all automata with $n > 7$ will be bisimilar to $n = 4$ or one of them could not be a better solution, however, in this case closer analysis of the remaining four glitches would show that they are real non-determinism that cannot be resolved with the addition of more states.

In this section, we answered RQ 2, outlining the practical considerations that should be taken into account when applying PMSAT and demonstrating the use of the guidelines on an example.

## 5 IMPLEMENTATION AND EVALUATION

In this section, we present an implementation of the PMSAT inference algorithm defined in Section 3 and evaluate its performance in terms of its ability to infer the correct model from faulty data and its computation speed based on the number of states, length of the trace, number of faults and types of faults. We also compare PMSAT to the IO ALERGIA [30] algorithm as a baseline. Additionally, we present three use cases to evaluate the algorithm in practice using the considerations outlined in Section 4.

The evaluation is based on our implementation in Python [53][1], which uses **AALpy** [38], an automata learning library used for the handling, visualisation and generation of Moore machines and its implementations of L$^*$[2] and random walks for the generation of our test traces, as well as **PySAT** [20], a Python interface for a variety of SAT solvers as well as different PMSAT implementations.

### 5.1 PMSAT Inference Performance

*5.1.1 Benchmarking Setup.* To test the performance and accuracy of the PMSAT inference algorithm presented in Section 3, we used the following setup: First, we generated ten random Moore machines with $|I| = 3$ from a given number of states $n$ and output alphabet size $|O|$. Next, we generated traces representative of each Moore machine by running the L$^*$ algorithm [2] and recording the observed inputs and outputs. Finally, we discarded a certain percentage of steps, i.e., input-output pairs, randomly from the traces to introduce non-determinism and simulate message loss or a similar type of fault. We varied the size of the output alphabet $|O| \in [2, 9]$, the number of states $n \in [|O|, 17]$ and the percentage of discarded steps $g \in \{0\%, 1\%, 2\%, 5\%, 10\%\}$ for a total of 500 combinations. For these parameters and Moore machines the longest traces produced had 2300 steps. The CNF formulas for these automata had between 450 and one million hard clauses, with a median of 105054, and between 200 and 50000 variables, with a median of 8225.

We then attempted to infer an automaton from such mutated traces, which we refer to as an *experiment*. The experiments were performed with the *correct number of states as its input* and given a

[1]The algorithm is maintained at https://gitlab.com/felixwallner/pmsat-inference

**Table 3: Percentage of the 400 experiments per number of states that ran into the one hour timeout.**

| $n$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|
| timeout % | 0.6 | 0.3 | 0.5 | 1.5 | 3.5 | 5.8 | 10.3 | 17.5 | 27 | 33 |

*one hour time limit* each, running on an Ubuntu 22.02 system with an AMD Ryzen 9 5950X CPU. We recorded solve time and correctness, i.e., whether or not the correct automaton could be mined, along with some other metrics. An experiment counts as correctly inferred only if the resulting model is *bisimilar* to the original and the result could be calculated in the one hour time limit. The solve time for the correct number of states is a significant metric insofar as the inference algorithm can be run for any number of different $n$ on the same set of traces in parallel because all inferences are mutually independent. Therefore, we only ran the algorithm for the correct number of states to save time during testing.

*5.1.2 Partial Max-SAT Implementations.* There are a variety of implementations of the Partial Max-SAT algorithm: Morgado et al. [35] give an overview of different iterative and core-guided algorithms.

First, we evaluated the performance of three different Partial Max-SAT implementations in PySAT [20], namely the Linear SAT-UNSAT (LSU) [32, 35], Fu and Malik (FM) [14, 29] and relaxable cardinality constraints (RC2) [21, 34, 36] algorithms. As RC2 performed best in this initial screening we chose it for the rest of our benchmarks. RC2 is core-guided and has to perform number-of-glitches-many SAT calls until a satisfiable solution is found.

As a practical consideration we mention here LSU, which has the advantage over RC2 that it can be interrupted and still result in a satisfiable, if sub-optimal, solution. This could be useful if the solver is given a time budget and a sub-optimal solution is acceptable.

*5.1.3 Evaluation.* Here we evaluate the following points:

**Correct Inference and Runtime.** We evaluate the percentage of correctly inferred models and runtime of PMSAT in Figures 3a, 3b, and 3c: The first two figures show the percentage of correct inferences depending on the output alphabet size and the number of dropped steps, respectively, while the last figure shows the average solve time for the latter. Timing out after one hour counts both towards incorrect inferences as well as towards the mean runtime. We show the percentage of timeouts separately in Table 3 as the timely computation of our results is an important aspect of the inference algorithm. No timeouts occurred for $n \leq 7$.

In Figure 3a we can see that the larger the difference between $|O|$ and $n$, the worse performance gets. This can in part be explained by Equation 1 because all outputs can be paired with every state and our optimisation in Equation 8 only fixes the first $|O|$ state outputs. The even larger influence on correct inferences and solve time is the amount of faults in the traces, which can be seen in Figures 3b and 3c. While not every step discarded from the trace will result in a glitch, most of them do. They impact the runtime insofar as RC2 has to perform as many calls to the SAT solver as there are glitches to find the maximum number of satisfiable soft clauses.

The number of states $n$ with which to infer the automata also has an impact on runtime. In Section 3.2, we showed that the number of clauses grows quadratically with the number of states. Even with glitch-free traces PMSAT started running into timeouts with $n \geq 17$, which can be seen in Figure 3b. This would of course mean that traces with glitches, which would require more than a single SAT call, would have taken even longer to calculate.

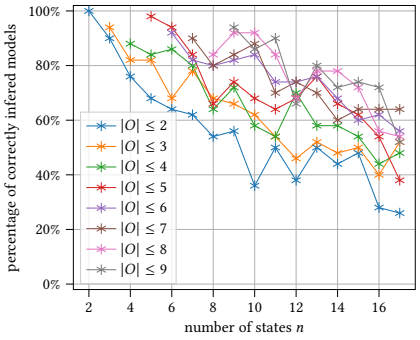
To summarise, the algorithm solves fastest and most reliably with a low number of faults in the traces due to the multiple SAT calls necessary for RC2 to find the best possible solution as well as with a high number of different outputs and small $n$.

**Runtime wrt. Trace Steps.** The runtime with respect to the length of the trace for 100 randomly generated automata with $n = 8$, $|O| \leq 5$ and 1% discarded trace steps is presented in Figure 3e. Similarly to our other experiments, the representative traces were first learned with $L^*$, then 12 different experiments were performed where the traces were extended by up to 5500 steps using random walks for a total of 1200 experiments. The box plots in Figure 3e show the solving time for these experiments, collecting traces into buckets with 1000 steps each. Each box encompasses the first quartile ($Q_1$) to third quartile ($Q_3$) and the whiskers are plotted at 1.5 times the interquartile range ($Q_3 - Q_1$) while the points are outliers. As can be seen from the figure, for these parameters the solving time was seconds for traces up to 3000 steps, minutes for up to 6000 steps and hours for larger traces. More than half of all traces longer than 6000 steps timed out after one hour while not a single timeout occurred for traces shorter than 1000 steps.

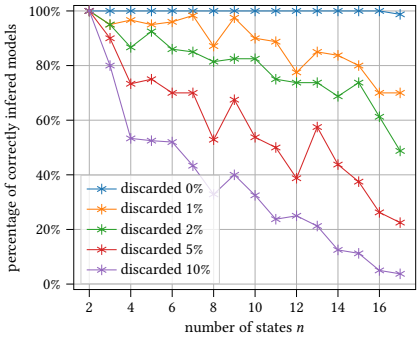
**Different Fault Types.** Our benchmarks focus on discarded trace steps as fault type due to it translating to a very common real world problem, namely message loss. This fault type fulfils both requirements of the type of noise our algorithm can deal with: *sparsity* and *uniform distribution* across the entire trace. We evaluated three additional fault types, which fulfil these requirements: *duplicating* random steps, i.e., input-output pairs, which models a message being sent multiple times, *inserting* random steps, which could be due to interleaving messages between different actors, and *swapping* two neighbouring steps, which could appear due to timing issues in the transmission, e.g., congestion. Correctness (excluding timeouts) for all four fault types is similar. The major difference between them is in how many glitches they produce in the algorithm: A single discarded step induces at most one, an inserted or duplicated step at most two, and two swapped steps at most three glitches. This is because each transition to or from an inserted or swapped step may be a separate glitch as our definition of glitches is based on transitions, which explains the difference in performance in Figure 3d. Therefore, while our algorithm can deal with other fault types, the fault type can have an impact on performance due to our algorithm's glitch representation.

**Comparison to IO ALERGIA.** Finally, we compare our approach to the IO ALERGIA [30] algorithm as a baseline. It is implemented in AALpy [38] and was recently used [8, 37, 48]. The reasons why we chose IO ALERGIA for our comparison were the following:

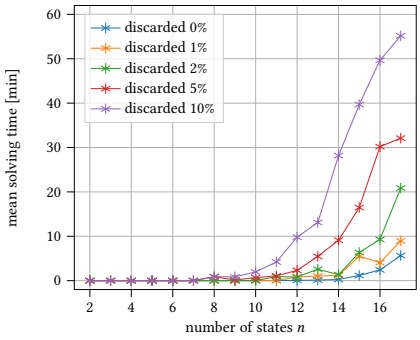
(1) We wanted to compare to an algorithm that does not require interaction with the system but is able to work with pre-existing traces. This rules out active inference algorithms, like [1], [18] and [43].



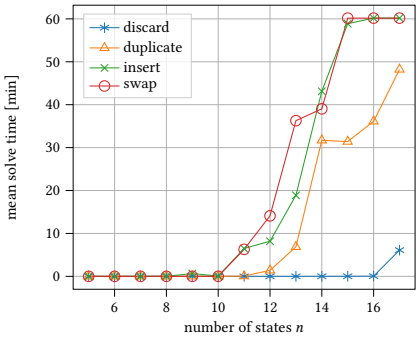

**(a) Percentage of correctly inferred models based on $n$ and the output alphabet size, 50 experiments over all different percentages of discarded trace steps.**



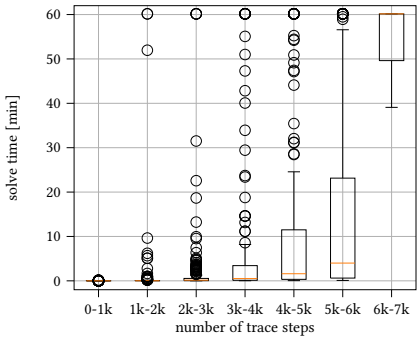

**(b) Percentage of correctly inferred models based on $n$ and the percentage of discarded trace steps, 80 experiments over all output alphabet sizes.**



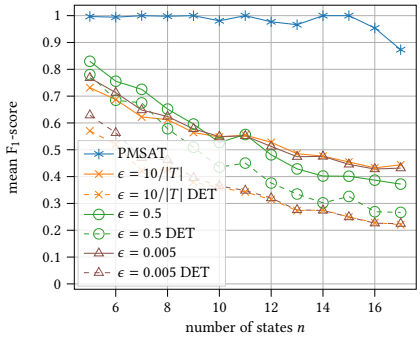

**(c) Average solving time based on $n$ and the percentage of discarded trace steps, 80 experiments over all output alphabet sizes**




**(d) Average solving time based on $n$ and different types of faults all with 1% of steps randomly affected, for 10 random automata each.**




**(e) Solving time based on number of trace steps for $n = 8$, $|O| \leq 5$ and 1% discarded steps for 100 random automata, each trace extended with random walks.**




**(f) Average $F_1$-score for 10 random automata per number of states with $|O| \leq 5$ and 1% discarded trace steps for PMSAT and IO ALERGIA with different $\epsilon$.**

**Figure 3: Different benchmark results on randomly generated Moore machines**

(2) The algorithm should be able to work directly with noise so that both algorithm could be compared using the same noisy traces. This rules out algorithms that cannot cope with non-deterministic data (traces), like [41] and [15].
(3) The algorithm should support input-output behaviour directly. Particularly, the resulting model should be a Moore machine or at least a closely related formalism to enable a sensible comparison. This rules out algorithms that produce different model types like [49], [52], [31] and [12].

We are not aware of an algorithm that fulfils all of these points explicitly, however, a close enough match to Moore machines would be Markov-decision processes, which are equivalent to our stochastic Moore machines. We chose IO ALERGIA because a mature implementation was readily available at that time. We consider a future comparison with MDP-BW [8] implemented in Jajapy [44]. Further discussion on different algorithms that were considered for comparison can be found in the related work in Section 6.

Note that we can interpret our PMSAT-based approach as first learning a stochastic Moore machine and then extracting a (deterministic) Moore machine by removing the transitions with the lowest frequencies/probabilities if multiple transitions on the same input from the same state exist. IO ALERGIA can also be adapted to produce a (deterministic) Moore machine in this way. We call the algorithms *stochastic* and *deterministic* to differentiate between the produced model types. We evaluate different values for the parameter $\epsilon$, which configures the significance level of statistical tests for difference of candidate states, where lower levels lead to smaller automata. In Figure 3f we compare three automata, namely the deterministic PMSAT, stochastic IO ALERGIA and deterministic IO ALERGIA with the deterministic ground truth automaton via conformance testing: We first sample 10000 random traces via random walks from both the ground truth and from the learned automaton. Then we compute the fraction of samples that can be produced by the other automaton respectively. These fractions are the precision and recall for the learned automata w.r.t. the ground truth. Finally, the harmonic mean between these two fractions is called the $F_1$-score [24]. This approach compares language similarity, with scores closer to one meaning the languages are more similar, while also allowing the comparison between stochastic and deterministic automata. As can be seen in Figure 3f, PMSAT performs vastly better in this respect than both regular IO ALERGIA and its simple deterministic variant.

**Optimisation.** Without Equation 8 the total solve time for evaluating the entire benchmarking set takes about 753 hours instead of 500 hours, about 50% longer, and on average about 182 seconds longer per experiment with 250 additional timeouts.

## 5.2 Use Case Studies

We present three different use cases, the first two of which are automotive measurement devices, which have a different fault model in the form of time-triggered state changes, and a Bluetooth Low Energy device, which experiences message loss.

*5.2.1 Automotive Measurement Devices.* A class of devices especially of interest are automotive measurement devices, which we want to model. We present two different ones here for which we both learned from *a single* representative trace because resetting either of them into their respective starting state would have taken a relatively long time. Both devices are designed to run continuously. After inferring solutions for different $n$ and choosing a result we verified our choice with domain experts. All traces and results can be found in full online [53]. The measurement devices have internal state variables that can be queried to find out the current internal state. However, some of the internal state labels appear *multiple times* for different states due to other unobservable variables. Otherwise, every state would have a unique output and the devices would be trivial to model. During testing we polled the internal state variable of the devices on average every 0.12 seconds

*Time-Triggered State Changes.* Both devices exhibit behaviours that induce faults into the traces, which we call *time-triggered* state changes. These are state changes that are initiated by the device without any external input and that are also not immediately obvious to an outside observer without additional state queries. For our purposes, we can think of these state changes as being triggered by a timer that starts running after a specific state is entered. This may or may not be the actual underlying mechanism of such state changes in practice. In both devices most states represent actions that the device can do, such as measuring, which automatically returns to the standby state once the action is completed. Usually, we will quickly explore a state and leave it via a transition, but if the device stays in a time-triggered state for too long or the timer is very short, then it may *appear* that the last input we used lead to a different state when in reality the device performed a time-triggered transition in-between external inputs. In order to better represent this behaviour we added an additional input, namely 'WAIT', which would wait until a time-triggered state change occurred or for a maximum of 5 minutes if no such state change is observed. Nevertheless input race conditions were still possible and resulted in the glitches we observed in our traces.

*Advanced Particle Counter.* The *AVL Advanced Particle Counter* (APC) is a device to measure the number of solid particles in a stream of exhaust gas through laser scattering on individual particles [6]. The model of the APC was inferred from a single trace with 385 steps, $|I| = 5$ and $|O| = 7$ for which the results are shown in Table 4. Each of these PMSAT solves took less than a second individually.

We can see that until $n = 9$ the number of glitches falls considerably for such a short trace. Then $n = 10$ does not remove any glitches and as such is not of interest, while $n = 11$ does remove a glitch again. The automata with $n = 12$ and $n = 13$ both have a large number of transitions only taken once, which might be special cases encoded into the automaton, and have a number of non-input-complete states. Choosing between $n = 9$ and $n = 11$ is more tricky. We decided on $n = 9$ due to its more general behaviour and it being input-complete, which turned out to be correct.

**Table 4: Statistics of inferring the APC with different $n$ as parameter, with $n_{reach}$ dominant reachable states, number of glitches and different statistics of frequencies (fr.) for glitched $\delta_g$ and dominant $\delta$ transitions.**

| $n$ | $n_{reach}$ | # Glitches | Mean $\delta_g$ fr. | Max $\delta_g$ fr. | Min $\delta$ fr. |
|---|---|---|---|---|---|
| 7 | 7 | 12 | 3 | 6 | 4 |
| 8 | 8 | 6 | 2 | 4 | 4 |
| 9 | 9 | 2 | 1 | 1 | 1 |
| 10 | 10 | 2 | 1 | 1 | 0 |
| 11 | 11 | 1 | 1 | 1 | 0 |
| 12 | 12 | 1 | 1 | 1 | 0 |
| 13 | 13 | 0 | 0 | 0 | 0 |

**Table 5: Statistics of inferring the Smoke Meter with different $n$ as parameter, with $n_{reach}$ dominant reachable states, number of glitches and different statistics of frequencies (fr.) for glitched $\delta_g$ and dominant $\delta$ transitions.**

| $n$ | $n_{reach}$ | # Glitches | Mean $\delta_g$ fr. | Max $\delta_g$ fr. | Min $\delta$ fr. |
|---|---|---|---|---|---|
| 9 | 8 | 52 | 17.33 | 25 | 3 |
| 10 | 8 | 27 | 13.5 | 23 | 3 |
| 11 | 11 | 4 | 4 | 4 | 3 |
| 12 | 12 | 4 | 4 | 4 | 0 |
| 13 | 13 | 1 | 1 | 1 | 0 |
| 14 | 14 | 0 | 0 | 0 | 0 |

*Smoke Meter.* The *AVL Smoke Meter* is an automotive measurement device to measure the amount of smoke in a stream of exhaust gas through the optical blackening of a filter paper [7]. Table 5 shows the learning process of the Smoke Meter using the guidelines described in Section 4 from a single trace with 789 steps, $|I| = 6$ and $|O| = 9$. Individually, PMSAT runs took less than a second.

Interestingly, the automata with $n = 9$ and $n = 10$ only had eight reachable states and were bisimilar to each other. Because we wanted to have all model outputs reachable, these two models would not be the final results. With $n = 11$ the glitches were reduces drastically, which also turned out to be the correctly inferred model.

*5.2.2 Bluetooth Low Energy.* One of the fault types we are especially interested in is message loss. In the case study on learning Bluetooth Low Energy (BLE) devices by Pferscher and Aichernig [43], they present an active learning approach for multiple BLE devices that regularly deals with message loss. From their case study, we selected the *Nordic nRF52832 RF System on a Chip* [40] for our own use case for two reasons: Firstly, it has a relatively small state space and secondly it took the longest to learn for the evaluated BLE devices with smaller state spaces. The correctly learned automaton and the software to interact with the BLE devices is available in the repository [42] connected with their case study. From this nRF52832 automaton, we dropped a single input due to the device having 27 (Moore) states originally in order to make it

**Table 6: Statistics of inferring the nRF52832 BLE chip with different $n$ as parameter, with $n_{reach}$ dominant reachable states, number of glitches and different statistics of frequencies (fr.) for glitched $\delta_g$ and dominant $\delta$ transitions.**

| $n$ | $n_{reach}$ | # Glitches | Mean $\delta_g$ fr. | Max $\delta_g$ fr. | Min $\delta$ fr. |
|---|---|---|---|---|---|
| 9 | 9 | 106 | 8.83 | 16 | 7 |
| 10 | 10 | 40 | 5.71 | 10 | 7 |
| 11 | 11 | 33 | 4.71 | 10 | 2 |
| 12 | 12 | 25 | 3.13 | 11 | 3 |
| 13 | 13 | 14 | 2 | 8 | 2 |
| 14 | 14 | 2 | 1 | 1 | 2 |
| 15 | 14 | 1 | 1 | 1 | 0 |
| 16 | 15 | 1 | 1 | 1 | 0 |

better comparable to the sizes of the other use case studies. We discuss the algorithm's performance for larger state spaces above.

We then used the W-method by Chow [11] and Vasilevskii [50] to generate a set of 224 representative input sequences, which we extended to length ten each with random inputs. We then *replayed* these input sequences on the real nRF52832 BLE device, which led to packet loss in the thus generated traces with a total of 2240 steps. It took about three hours to replay all 224 traces on the device. Finally, we inferred automata from these generated device traces, for which the results can be found in Table 6, with $|I| = 8$ and $|O| = 9$. Each of these PMSAT runs took less than ten seconds individually. As can be seen up to and including $n = 13$, there is a high frequency of glitches, which are encoded into the automaton with $n = 14$ states. The automaton with $n = 15$ is bisimilar to the previous one and all automata with $n > 15$ include at least one state that is not reachable via dominant transitions, as shown in the example in the previous section in Figure 2c. The automaton with $n = 14$ was correctly inferred.

In this section, we answered RQ 3, showing the performance of our approach in regards to different criteria on benchmarks in Section 5.1.3 and on three use case studies.

## 6 RELATED WORK

Inferring a minimal automaton with a given number of states and consistent with a set of traces was shown to be NP-hard by Gold [16]. Nevertheless, there exists a variety of inference approaches for different types of automata using, among others, CSP, SAT and SMT solving: The problem was originally stated by Biermann and Feldman [9] as a CSP problem, which was later formulated as a SAT problem by Grinchtein et al. [17]. Their approach was improved by Heule and Verwer [19] by combining exact SAT solving with greedy state-merging and heuristics for model inference. Avellaneda and Petrenko [5] present an incremental approach especially for inferring automata from long traces. Neider [39] proposes a technique to infer deterministic finite automata (DFAs) with a fixed number of states. Smetsers et al. [46] infer DFAs, Mealy machines, and register automata from observed behaviour using SMT solving. Tappler et al. [47] uses SMT solving to learn timed automata.

Some inference algorithms do not rely on SAT or SMT solvers, such as RPNI [41], which uses state merging to build a deterministic automaton from a set of traces. Giantamidis et al. [15] formalise the problem of learning Moore machines from traces. The hW-inference algorithm [18] can infer a model from a single trace instead of requiring many samples. Luo et al. [28] present a distributed scalable algorithm, while Busani and Maoz [10] present an approach that allows for the sampling of traces with statistical guarantees for better scalability.

All of these approaches assume that the data accurately represents the underlying FSM and does not contain non-deterministic faults. There is a variety of inference methods for practical settings that deal with non-deterministic behaviour:

One approach is to use *active* inference where the system under learning can be actively queried to refine the inferred model. This has the advantage of allowing multiple executions of the same queries if inconsistencies are detected, but it requires the system to be available, which is often not the case if data is taken from logs or records. One such approach by Aichernig et al. [1] masks detected non-determinism with sink states, which then allows them to use a regular deterministic learning algorithm, such as L* [2], to infer the model. Another approach by Pferscher and Aichernig [43] uses L* to learn BLE devices. They experience message losses, connection errors and message delays, which they deal with by repeating queries multiple times and discarding outlier traces.

An alternative approach is to model the underlying system differently to include this non-determinism. The downside of such an approach is that the noise appears in the structure of the FSMs, which is often not desirable, especially if the underlying system is known to be deterministic. IO ALERGIA [30] uses state merging to build stochastic finite automata from a set of traces. This allows it to learn even if the traces are non-deterministic, however, the resulting automaton includes the non-deterministic behaviours. Bacci et al. [8] improve on the model quality by obtaining a baseline model of a Markov-decision process (e.g. from IO ALERGIA) and iteratively updating its transition probabilities. Emam and Miller [12] infer extended probabilistic FSMs. Vazquez de Parga et al. [51] present a family of inference algorithms for non-deterministic finite automata (NFAs), while Lardeux and Monfroy [26] formulate the problem of learning an NFA of a certain size from data as a SAT problem. While these methods can deal with the noise in our data, they encode it directly into the inferred automaton and may require postprocessing to remove it.

Some inference methods for deterministic models take noise directly into account: Angluin and Laird [3] researched random noise and how to compensate for it, Kearns [22] dealt with noise in probabilistic learning, while Khmelnitsky et al. [23] researched robustness of L* with regards to learning DFAs from data with random and structured noise. Sebban and Janodet [45] extended the RPNI algorithm [41] by relaxing the state merging rules, to infer DFAs from noisy data. Lucas and Reynolds [27] use an evolutionary method for learning DFAs from noisy and noiseless data. Ulyantsev et al. [49] mine DFAs from both noisy and noiseless data. They use regular SAT solving and a different fault model that requires a number of additional clauses not found in our own algorithm while we leverage PMSAT to formulate the problem concisely. They define an upper bound on the number of glitches, among other parameters, and iteratively perform calls to the SAT solver to determine the number of states of the automaton.

The MINT framework [52] and GK-tail+ [31] are able to mine models from the richer class of deterministic extended FSMs from traces consisting of events parametrized by data. MINT learns guards in terms of parameters as part of the state machine by using state merging. Furthermore, it infers data classifiers that are used to resolve non-determinism during inference, which may address inflexibilities such as noise or spurious events. GK-tail+ infers constraints on a per-transition basis. However, both methods do not distinguish between input and output symbols like they are used in Moore machines.

In conclusion, there exists a variety of model mining algorithms, however, most differ in one of the following aspects: (1) Some cannot handle noise and need pre-processing, which may require domain knowledge. (2) Others learn non-deterministically or stochastically and encode the noise directly into the model, which we seek to avoid. (3) Some are able to deal with noise but require active interaction with the system instead of learning from traces. (4) Finally, some mine different types of models instead of Moore machines. Thus our approach combines a set of specific capabilities in a novel way.

To the best of our knowledge the method presented in this paper is the first using PMSAT for model inference from noisy data.

## 7 THREATS TO VALIDITY AND LIMITATIONS

We identified the following threats to the validity of our work:

**Encoding of Glitches.** Our PMSAT encoding assumes glitches to be transitions that are not present in the underlying automatons $\delta_{\mathcal{M}}$. While we cannot guarantee that this encoding will suffice to model all types of real-world glitches, it did work very well for the different fault types encountered in our benchmarks and use cases.

**Systematic Faults.** Our algorithm targets sporadic, i.e., transient, and randomly distributed noise. This noise should not be included in the model due to it not being part of the system behaviour. Contrary, systematic faults, i.e., non-trivial system behaviour such as exceptions among others, will be modelled if it appears often enough in the data. This is not only expected but also wished for as model mining is often used to discover unexpected systematic faults or insecure system behaviour. The inclusion of such systematic faults in the model is common to all model mining algorithms.

In the use case studies about the measurement devices, the underlying faults of the devices were systematic, i.e., the time-triggered state changes, which happen only in certain states, however, due to the abstraction of time the effects appear random (enough) in the data for our algorithm to learn the correct models.

**Biased Benchmarks.** A possible threat to validity is that our algorithm works only on selected systems or that our evaluation was biased. To mitigate this bias we exclusively used randomly generated automata in our benchmarks. Additionally, we provided three practical use case studies from two very different domains: communication protocols with BLE, and measurement devices with the APC and Smoke Meter devices.

**Scalability.** The primary limitation of the algorithm is scalability, due to the NP-hard nature of (partial) Max-SAT [25]. While the algorithm might not be able to solve problems with hundreds of states or millions of trace steps in reasonable time, we demonstrated its capabilities for small to moderately large automata/traces (up to 17 states or 6000 trace steps), which are sufficient to learn industrial real-world use cases like measurement devices or communication protocols.

## 8 CONCLUSION

We presented a novel approach based on Partial Max-SAT to infer deterministic Moore machines from noisy data with a certain number of states. The approach is able to work with a single execution trace or with a set of traces and will find a deterministic model consistent with as many observations as possible. We further presented practical considerations regarding the best choice of the size of the automaton and evaluated our method's performance on a set of randomly generated Moore machines with different parameters. The preliminary experimental results show that the approach can process traces up to 17 states or up to 6000 steps within few minutes. However, half of all our experiments with 17 states or longer than 6000 traces timed out after one hour. Finally, we showed that the method can deal with different fault types in data by means of three use cases: Two industrial measurement devices, each inferred from a single execution trace that experienced time-triggered state changes, and one BLE device that experienced message loss. The method is expected to be used for software and firmware regression testing and serves as a basis for device simulation for digital twins.

Our future work may include direct comparison between this approach and other state-of-the-art methods like the MDP-BW algorithm [8, 44], as well as more extensive performance evaluation against established benchmark automata.

## ACKNOWLEDGMENTS

This work was a collaboration between AVL List GmbH and Graz University of Technology in the LearnTwins project funded by the Austrian Research Promotion Agency (FFG) under grant 880852. We would like to thank Andrea Pferscher for providing the data for the BLE use case and Benjamin von Berg for the fruitful discussions. We also would like to thank the anonymous reviewers for their valuable feedback and insightful comments.

## REFERENCES

[1] Bernhard K. Aichernig, Christian Burghard, and Robert Korosec. 2019. Learning-Based Testing of an Industrial Measurement Device. In *NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11460)*, Julia M. Badger and Kristin Yvonne Rozier (Eds.). Springer, 1–18. https://doi.org/10.1007/978-3-030-20652-9_1

[2] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Information and Computation* 75, 2 (1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

[3] Dana Angluin and Philip D. Laird. 1987. Learning From Noisy Examples. *Mach. Learn.* 2, 4 (1987), 343–370. https://doi.org/10.1007/BF00116829

[4] Josep Argelich and Felip Manyà. 2007. Partial Max-SAT Solvers with Clause Learning. In *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4501)*, João Marques-Silva and Karem A. Sakallah (Eds.). Springer, 28–40. https://doi.org/10.1007/978-3-540-72788-0_7

[5] Florent Avellaneda and Alexandre Petrenko. 2018. FSM Inference from Long Traces. In *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10951)*, Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik P. de Vink (Eds.). Springer, 93–109. https://doi.org/10.1007/978-3-319-95582-7_6

[6] AVL List GmbH. Nov. 2019. *AVL 489 Particle Counter - product guide.*

[7] AVL List GmbH. Sep. 2013. *AVL 415SE Smoke Meter - product guide.*

[8] Giovanni Bacci, Anna Ingólfsdóttir, Kim G. Larsen, and Raphaël Reynouard. 2021. Active Learning of Markov Decision Processes using Baum-Welch algorithm. In

*20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021, Pasadena, CA, USA, December 13-16, 2021*, M. Arif Wani, Ishwar K. Sethi, Weisong Shi, Guangzhi Qu, Daniela Stan Raicu, and Ruoming Jin (Eds.). IEEE, 1203–1208. https://doi.org/10.1109/ICMLA52953.2021.00195

[9] Alan W. Biermann and Jerome A. Feldman. 1972. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Trans. Computers* 21, 6 (1972), 592–597. https://doi.org/10.1109/TC.1972.5009015

[10] Nimrod Busany, Shahar Maoz, and Yehonatan Yulazari. 2019. Size and Accuracy in Model Inference. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 887–898. https://doi.org/10.1109/ASE.2019.00087

[11] Tsun S. Chow. 1978. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering* 4, 3 (1978), 178–187. https://doi.org/10.1109/TSE.1978.231496

[12] Seyedeh Sepideh Emam and James Miller. 2018. Inferring Extended Probabilistic Finite-State Automaton Models from Software Executions. *ACM Trans. Softw. Eng. Methodol.* 27, 1 (2018), 4:1–4:39. https://doi.org/10.1145/3196883

[13] Jean-Claude Fernandez and Laurent Mounier. 1991. "On the Fly" Verification of Behavioural Equivalences and Preorders. In *Computer Aided Verification, 3rd International Workshop, CAV '91, Aalborg, Denmark, July, 1-4, 1991, Proceedings (Lecture Notes in Computer Science, Vol. 575)*, Kim Guldstrand Larsen and Arne Skou (Eds.). Springer, 181–191. https://doi.org/10.1007/3-540-55179-4_18

[14] Zhaohui Fu and Sharad Malik. 2006. On Solving the Partial MAX-SAT Problem. In *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4121)*, Armin Biere and Carla P. Gomes (Eds.). Springer, 252–265. https://doi.org/10.1007/11814948_25

[15] Georgios Giantamidis, Stavros Tripakis, and Stylianos Basagiannis. 2021. Learning Moore machines from input-output traces. *Int. J. Softw. Tools Technol. Transf.* 23, 1 (2021), 1–29. https://doi.org/10.1007/s10009-019-00544-0

[16] E. Mark Gold. 1978. Complexity of Automaton Identification from Given Data. *Inf. Control.* 37, 3 (1978), 302–320. https://doi.org/10.1016/S0019-9958(78)90562-4

[17] Olga Grinchtein, Martin Leucker, and Nir Piterman. 2006. Inferring Network Invariants Automatically. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4130)*, Ulrich Furbach and Natarajan Shankar (Eds.). Springer, 483–497. https://doi.org/10.1007/11814771_40

[18] Roland Groz, Nicolas Brémond, Adenilso Simao, and Catherine Oriat. 2020. hW-inference: A heuristic approach to retrieve models through black box testing. *Journal of Systems and Software* 159 (2020), 110426. https://doi.org/10.1016/j.jss.2019.110426

[19] Marijn Heule and Sicco Verwer. 2013. Software model synthesis using satisfiability solvers. *Empir. Softw. Eng.* 18, 4 (2013), 825–856. https://doi.org/10.1007/s10664-012-9222-z

[20] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*. 428–437. https://doi.org/10.1007/978-3-319-94144-8_26

[21] Alexey Ignatiev, António Morgado, and João Marques-Silva. 2019. RC2: an Efficient MaxSAT Solver. *J. Satisf. Boolean Model. Comput.* 11, 1 (2019), 53–64. https://doi.org/10.3233/SAT190116

[22] Michael J. Kearns. 1998. Efficient Noise-Tolerant Learning from Statistical Queries. *J. ACM* 45, 6 (1998), 983–1006. https://doi.org/10.1145/293347.293351

[23] Igor Khmelnitsky, Serge Haddad, Lina Ye, Benoît Barbot, Benedikt Bollig, Martin Leucker, Daniel Neider, and Rajarshi Roy. 2022. Analyzing Robustness of Angluin's L* Algorithm in Presence of Noise. In *Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022 (EPTCS, Vol. 370)*, Pierre Ganty and Dario Della Monica (Eds.). 81–96. https://doi.org/10.4204/EPTCS.370.6

[24] Iraklis A. Klampanos. 2009. Manning Christopher, Prabhakar Raghavan, Hinrich Schütze: Introduction to information retrieval. *Inf. Retr.* 12, 5 (2009), 609–612. https://doi.org/10.1007/s10791-009-9096-x

[25] Mark W. Krentel. 1988. The Complexity of Optimization Problems. *J. Comput. Syst. Sci.* 36, 3 (1988), 490–509. https://doi.org/10.1016/0022-0000(88)90039-6

[26] Frédéric Lardeux and Éric Monfroy. 2021. Improved SAT Models for NFA Learning. In *Optimization and Learning - 4th International Conference, OLA 2021, Catania, Italy, June 21-23, 2021, Proceedings (Communications in Computer and Information Science, Vol. 1443)*, Bernabé Dorronsoro, Lionel Amodeo, Mario Pavone, and Patricia Ruiz (Eds.). Springer, 267–279. https://doi.org/10.1007/978-3-030-85672-4_20

[27] Simon M. Lucas and T. Jeff Reynolds. 2005. Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.* 27, 7 (2005), 1063–1074. https://doi.org/10.1109/TPAMI.2005.143

[28] Chen Luo, Fei He, and Carlo Ghezzi. 2017. Inferring software behavioral models with MapReduce. *Sci. Comput. Program.* 145 (2017), 13–36. https://doi.org/10.1016/j.scico.2017.04.004

[29] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. 2009. Algorithms for Weighted Boolean Optimization. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5584)*, Oliver Kullmann (Ed.). Springer, 495–508. https://doi.org/10.1007/978-3-642-02777-2_45

[30] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. 2016. Learning deterministic probabilistic automata from a model checking perspective. *Mach. Learn.* 105, 2 (2016), 255–299. https://doi.org/10.1007/s10994-016-5565-9

[31] Leonardo Mariani, Mauro Pezzè, and Mauro Santoro. 2017. GK-Tail+ An Efficient Approach to Learn Software Models. *IEEE Trans. Software Eng.* 43, 8 (2017), 715–738. https://doi.org/10.1109/TSE.2016.2623623

[32] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. 2014. Incremental Cardinality Constraints for MaxSAT. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8656)*, Barry O'Sullivan (Ed.). Springer, 531–548. https://doi.org/10.1007/978-3-319-10428-7_39

[33] Edward F. Moore. 2016. Gedanken-experiments on sequential machines. In *Automata Studies. (AM-34), Volume 34*, C. E. Shannon and J. McCarthy (Eds.). Princeton University Press, 129–154. https://doi.org/doi:10.1515/9781400882618-006

[34] António Morgado, Carmine Dodaro, and João Marques-Silva. 2014. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8656)*, Barry O'Sullivan (Ed.). Springer, 564–573. https://doi.org/10.1007/978-3-319-10428-7_41

[35] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints An Int. J.* 18, 4 (2013), 478–534. https://doi.org/10.1007/s10601-013-9146-2

[36] António Morgado, Alexey Ignatiev, and João Marques-Silva. 2014. MSCG: Robust Core-Guided MaxSAT Solving. *J. Satisf. Boolean Model. Comput.* 9, 1 (2014), 129–134. https://doi.org/10.3233/sat190105

[37] Edi Muskardin, Martin Tappler, Bernhard K. Aichernig, and Ingo Pill. 2022. Reinforcement Learning under Partial Observability Guided by Learned Environment Models. *CoRR* abs/2206.11708 (2022). https://doi.org/10.48550/arXiv.2206.11708 arXiv:2206.11708

[38] Edi Muškardin, Bernhard Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. 2022. AALpy: an active automata learning library. *Innovations in Systems and Software Engineering* 18 (03 2022), 1–10. https://doi.org/10.1007/s11334-022-00449-3

[39] Daniel Neider. 2012. Computing Minimal Separating DFAs and Regular Invariants Using SAT and SMT Solvers. In *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7561)*, Supratik Chakraborty and Madhavan Mukund (Eds.). Springer, 354–369. https://doi.org/10.1007/978-3-642-33386-6_28

[40] Nordic Semiconductor. Nov. 2021. *nRF52832 Product Specification v1.8.*

[41] Jose Oncina and Pedro Garcia. 1992. Identifying Regular Languages In Polynomial Time. In *Advances in Structual and Syntactic Pattern Recognition, Volume 5 of Series in Machine Perception and Artificial Intelligence*. World Scientific, 99–108.

[42] Andrea Pferscher and Bernhard K. Aichernig. 2021. ble-learning: Fingerprinting Bluetooth Low Energy via Active Automata Learning. https://github.com/apferscher/ble-learning, accessed on March 14, 2023.

[43] Andrea Pferscher and Bernhard K. Aichernig. 2021. Fingerprinting Bluetooth Low Energy Devices via Active Automata Learning. In *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 13047)*, Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan (Eds.). Springer, 524–542. https://doi.org/10.1007/978-3-030-90870-6_28

[44] Raphaël Reynouard, Anna Ingólfsdóttir, and Giovanni Bacci. 2023. Jajapy: a learning library for stochastic models. In *Quantitative Evaluation of Systems - 20th International Conference, QEST 2023, Antwerp, Belgium, September 18-23, 2023, Proceedings (Lecture Notes in Computer Science)*, Nils Jansen Mirco Tribastone (Ed.). Springer.

[45] Marc Sebban and Jean-Christophe Janodet. 2003. On State Merging in Grammatical Inference: A Statistical Approach for Dealing with Noisy Data. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, Tom Fawcett and Nina Mishra (Eds.). AAAI Press, 688–695. http://www.aaai.org/Library/ICML/2003/icml03-090.php

[46] Rick Smetsers, Paul Fiterau-Brostean, and Frits W. Vaandrager. 2018. Model Learning as a Satisfiability Modulo Theories Problem. In *Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10792)*, Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira (Eds.). Springer, 182–194. https://doi.org/10.1007/978-3-319-77313-1_14

[47] Martin Tappler, Bernhard K. Aichernig, and Florian Lorber. 2022. Timed Automata Learning via SMT Solving. In *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13260)*, Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez (Eds.). Springer, 489–507. https://doi.org/10.1007/978-3-031-06773-0_26

[48] Martin Tappler, Edi Muskardin, Bernhard K. Aichernig, and Bettina Könighofer. 2023. Learning Environment Models with Continuous Stochastic Dynamics. *CoRR* abs/2306.17204 (2023). https://doi.org/10.48550/arXiv.2306.17204 arXiv:2306.17204

[49] Vladimir Ulyantsev, Ilya Zakirzyanov, and Anatoly Shalyto. 2015. BFS-Based Symmetry Breaking Predicates for DFA Identification. In *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 8977)*, Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe (Eds.). Springer, 611–622. https://doi.org/10.1007/978-3-319-15579-1_48

[50] M. P. Vasilevskii. 1973. Failure diagnosis of automata. *Cybernetics* 9, 4 (01 Jul 1973), 653–665. https://doi.org/10.1007/BF01068590

[51] Manuel Vázquez de Parga, Pedro García, and José Ruiz. 2006. A Family of Algorithms for Non Deterministic Regular Languages Inference. In *Implementation and Application of Automata, 11th International Conference, CIAA 2006, Taipei, Taiwan, August 21-23, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4094)*, Oscar H. Ibarra and Hsu-Chun Yen (Eds.). Springer, 265–274. https://doi.org/10.1007/11812128_25

[52] Neil Walkinshaw, Ramsay Taylor, and John Derrick. 2016. Inferring extended finite state machine models from software executions. *Empir. Softw. Eng.* 21, 3 (2016), 811–853. https://doi.org/10.1007/s10664-015-9367-7

[53] Felix Wallner, Bernhard K. Aichernig, and Christian Burghard. 2023. *PMSAT Inference Algorithm and Publication Artifacts.* https://doi.org/10.5281/zenodo.8341541