Automata Learning for Markov Decision Processes with Continuous Outputs

Benjamin von Berg BENJAMIN.VONBERG@TUGRAZ.AT
Graz University of Technology, Graz, Austria

Bernhard K. Aichernig BERNHARD.AICHERNIG@JKU.AT
Johannes Kepler Universität Linz, Linz, Austria

Editor: Editor's name

Abstract

Passive automata learning is a machine learning technique that infers an automaton model of a system from a set of execution traces. While originally conceived for inferring deterministic finite automata, many algorithms for different formalisms have since been developed, targeting systems with infinite state space as well as stochastic systems. However, the combined setting of stochastic reactive systems with continuous output domains is currently underexplored. For such systems, existing work typically applies classical methods on discretized data. In this paper, we present a method for lifting such models back to the continuous domain. The main contribution is a novel automata learning method that takes the continuous data into account during the inference process. This reduces the impact of the discretization method on the quality of the resulting models. We evaluate the approach by comparing it to established algorithms that operate exclusively on discretized data.

Keywords: Passive Automata Learning, Reactive Systems, Stochastic Systems, Cyber Physical Systems, Markov Decision Processes

1. Introduction

Computer systems are increasingly employed in systems that interact with the real world, often in a context that involves uncertainty. Many such Cyber-Physical Systems are safety critical and thus have to undergo rigorous verification. However, accurately modeling such systems is often costly or infeasible. This motivates the use of methods that can extract models from existing data, which can then be used for verification purposes. One such method is automata learning. However, most automata learning algorithms are only capable of inferring finite state systems and any applications thereof are in a purely finite setting. Extensions of automata learning to formalisms that support infinite or even continuous components – e.g. in the context of reactive systems – are an active field of research. In recent years, Gurung et al. (2023) developed methods for Hybrid / Cyber-Physical Systems. Foster et al. (2021) and Cassel et al. (2016) among others developed methods for Extended Finite State Machines. Where these methods are not applicable, a common approach is to (dynamically) abstract / discretize continuous data as done by Pellegrino et al. (2016) and Tappler et al. (2024). This reduces the problem to a finite setting at the cost of introducing discretization error. Discretization can also lead to apparent nondeterminism. However, continuous systems often have stochastic characteristics, e.g., due to measurement noise. This motivates the use of stochastic IO automata as is done by Tappler et al. (2024). A challenge that comes with this approach is how to choose the abstraction / discretization

© 2026 B. von Berg & B.K. Aichernig.

such that it is fine enough to capture relevant aspects of the system behavior but coarse enough to avoid degenerate data and computational complexity.

In this paper, we develop a passive automata learning method for stochastic IO automata with continuous outputs. The algorithm is based on the red-blue state-merging framework first presented by Lang et al. (1998). The method decreases the influence of the discretization method by using the continuous data during compatibility checking. An implementation of this method is available online[1]. It is based on the automata learning library AALpy by Muskardin et al. (2022), in particular its state-merging framework due to von Berg and Aichernig (2025).

The contributions of this work are as follows. (1) We present a general method for adding continuous output data back to a probabilistic model that was learned from discretized data. (2) We propose a passive automata learning algorithm that makes use of the continuous data during the learning process rather than using it only in postprocessing. (3) We evaluate our approach and compare it to other learning algorithms. (4) We provide an implementation of our methods based on an established automata learning library.

The remainder of the paper is structured as follows. Section 2 discusses the basic concepts required for the naive discretization-based approach. In Section 3, we provide the details of the post-processing method and our novel learning algorithm. This algorithm is compared to the naive approach in Section 4. Finally, Section 5 provides concluding remarks and relates to preexisting work.

2. Preliminaries

In this section, we first outline passive automata learning based on *state-merging*. This is done based on the algorithm IOAlergia by Mao et al. (2016) for inferring labeled Markov decision processes (MDPs)[2], which can be used to describe probabilistic systems with discrete behavior. We then discuss systems with continuous behavior and a simple abstraction mechanism. The latter introduces discretization error, but results in a discrete setting, where established automata learning algorithms can be applied to learn an approximation.

For any set S we denote the set of probability distributions over S as $\mathcal{D}(S)$, the set of all multisets composed of elements of S as $\mathcal{B}(S)$ and the powerset of S as $\mathcal{P}(S)$.

Markov Decision Processes. MDPs are a type of finite state automata that can be thought of as a probabilistic extension of Moore machines. We use the following definition, similar to Tappler et al. (2021).

Definition 1 *A **labeled Markov Decision Process** (MDP) is a tuple $\langle Q, q_0, I, O, L, \delta \rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, I and O are finite sets of inputs and outputs respectively, $L : Q \to O$ is an output function assigning each state a specific output and $\delta : Q \times I \rightharpoonup \mathcal{D}(Q)$ is a (partial) probabilistic transition function.*

The intuition behind δ is that for any state q, applying some input i results in a probabilistic transition to a state q' according to the respective distribution, i.e., $q' \sim \delta(q, i)$.

1. https://github.com/BenjaminVonBerg/Automata-Learning-for-Probabilistic-IO-Automata-with-Continuous-Outputs
2. We use the term MDP to stay consistent with the automata learning community, although the broader machine learning community usually refers to the same concept as partially observable MDP.

An MDP $\mathcal{M} = \langle Q, q_0, I, O, L, \delta \rangle$ is *input complete* iff δ is a total function, i.e., if the MDP accepts each input $i \in I$ in every state $q \in Q$.

An MDP $\mathcal{M} = \langle Q, q_0, I, O, L, \delta \rangle$ is *deterministic* iff, in any state q, the next state q' is fully determined by the applied input i and the observed output o. In other words, when applying i in q, there is at most one possible successor q' with output o. More formally, $\forall q \in Q, i \in I, o \in O : |\{q' \in Q \mid L(q') = o \wedge \delta(q, i)(q') \neq 0\}| \leq 1$. Determinism is a common assumption in automata learning algorithms and is also used throughout this paper. Note that the class of deterministic MDPs is less expressive than MDPs. For deterministic MDPs, the transition function δ can be lifted to input-output traces, such that $\delta(q, t)$ is equal to the unique state reached from q after applying / observing the trace t.

When inferring an MDP model from prerecorded data, the data is given as a multiset of traces. Each trace t is a finite sequence of symbols alternating between symbols of the output domain O and the set of inputs I. Additionally, each trace starts and ends with a system observation. Consequently, the set of all traces over I and O is defined as $\mathcal{TR}_{I,O} = O \cdot (I \cdot O)^*$. We omit the subscript if I and O are clear from the context.

State-Merging. Passive state-merging algorithms do not use MDPs directly, but rather keep track of how often each input-output pair has been observed, using an *empirical transition function* $\hat{\delta} : Q \times I \to Q \times \mathbb{N}$ instead of the distribution δ. A tuple $\langle Q, q_0, I, O, \hat{\delta} \rangle$ is called a Frequency IO Automaton (FIOA). Passive state-merging starts by constructing a so-called Prefix-Tree Automaton (PTA) from the available traces $T \subset \mathcal{TR}$. Each state in the PTA corresponds to a common prefix in the traces. This construction is possible only because of the assumption of determinism. Starting from the PTA, states are iteratively merged based on an algorithm-dependent compatibility criterion. Since the target MDP is assumed to be deterministic, each merge induces an equivalence relation on states. This corresponds to a partitioning π of states, i.e., $\pi \subset \mathcal{P}(Q)$ s.t. $\forall q \in Q : \exists! Q_k \in \pi : q \in Q_k$, from which a quotient automaton can be extracted. While the compatibility criterion has to operate on π, it commonly aggregates information from a compatibility criterion in terms of two individual states. When inferring probabilistic models, the compatibility criterion for two states typically compares the empirical distributions of outputs for each input using some statistical test. For example, Mao et al. (2016) use the Hoeffding bound. When several merges are possible, merge candidates can be ranked according to a score function. Finally, an MDP is extracted by normalizing frequencies to probabilities.

Usually, state merging algorithms do not guarantee that the resulting model is input complete. This can happen when insufficient data is available such that a specific input has never been observed for some state. A common method to ensure input completeness is to add self transitions for the missing state-input pairs in a post-processing step.

Systems with Continuous Behavior. Since this work is concerned with learning behavioral models of reactive systems with continuous outputs, MDPs as given in Definition 1 are not suitable. We make the following core assumptions about the system under learning, which essentially describe a version of MDPs with infinite state- and output-spaces.

1. The state space of the system is continuous and partially observable. The observations stem from a continuous domain $\mathcal{C}$, typically $\mathbb{R}^d$ for some $d \in \mathbb{N}_+$.
2. The system works in discrete time steps that process inputs from a finite set I.
3. The system exhibits stochastic behavior.

In the literature, *mappers* are commonly used to abstract from the complexity of continuous domains, e.g., by Pferscher and Aichernig (2020) or Aarts et al. (2015). In this work, we consider a simple version of this concept where a mapper is defined as a function $A : \mathcal{C} \to O$, where O – also called the (abstract) output alphabet – is a finite set of representatives of the continuous space, i.e., $O \subset \mathcal{C}$. We lift this function to traces and (multi-)sets of traces in the natural manner. This allows us write $A(T)$ for the multiset of *abstract traces* derived from a multiset T of *continuous traces* from the original system.

State-Merging for Systems with Continuous Behavior. The classic state-merging approach for learning probabilistic models can quickly degenerate if $N_\Sigma = |I| \cdot |O|$ is large compared to the number of samples if the system is not highly deterministic. This is because the amount of data per node in the PTA diminishes quickly with increasing depth as each node can have up to N_Σ children. As a consequence, the statistical compatibility tests performed during merging are supported by little data and thus not a reliable basis for decision-making. This reaches its extreme for continuous alphabets, where each node in the PTA except the root is expected to have only a single observation attached. Thus, employing state-merging on systems with continuous behavior is not feasible in a straightforward manner. However, we can learn an abstract MDP model $\mathcal{M} = \langle Q, q_0, I, O, L, \delta \rangle$ of a continuous system $\mathcal{S}$ from a collection of continuous traces $T \subset \mathcal{TR}_{I,\mathcal{C}}$ via a mapper $A : \mathcal{C} \to O$ by applying standard methods to a collection of abstract traces $A(T)$. An issue with this is that, in general, the abstraction $\mathcal{S}_A$ of $\mathcal{S}$ with respect to A can not be represented as a deterministic MDP (even if $\mathcal{S}$ is deterministic). Consequently, it is impossible to learn a correct model of the abstraction even in the limit. The focus of automata learning in this setting must therefore lie on learning a sufficiently accurate approximation of the system.

3. Methodology

3.1. Continuous Distributions via Postprocessing

In some cases, e.g., for generative models, it is desirable to not only map the outputs from the concrete to the abstract domain, but also the other way around. For an MDP $\mathcal{M} = \langle Q, q_0, I, O, L, \delta \rangle$ learned under some mapper A, instead of using a static representative for each discrete output $o \in O$, we can estimate a distribution of (continuous) outputs $\sigma(q) \in \mathcal{D}(\mathcal{C})$ for each state $q \in Q$. Together, the mapper, the learned model and the estimated continuous output distribution form a *continuous abstraction* $\mathcal{M}_c = \langle A, Q, q_0, I, O, L, \delta, \sigma \rangle$. To construct σ, we start by replaying the training data on $\mathcal{M}$ and recording the continuous outputs for each state. This results in a *continuous output map* $\hat{\sigma} : Q \to \mathcal{B}(\mathcal{C})$ from which the output distribution for each state can be estimated. In particular, we use Kernel Density Estimation with Gaussian kernels (compare Gramacki (2018)).

Given a continuous abstraction $\mathcal{M}_c = \langle A, Q, q_0, I, O, L, \delta, \sigma \rangle$, applying a fixed sequence of inputs $i = \langle i_1, \ldots, i_n \rangle$ for any $n \in \mathbb{N}$ results in a distribution $D_i \in \mathcal{D}(\mathcal{C}^{n+1})$ over sequences of the output space with length $n + 1$. A sample can be drawn from D_i using an iterative process in which a sequence of $n + 1$ states q_k and outputs x_k for $0 \leq k \leq n$ is sampled as follows. The first state in the sequence is the initial state q_0. In the k-th step, we sample a continuous output $x_k \sim \sigma(q_k)$. The continuous output x_k is discretized to obtain an abstract output $o_k = A(x_k)$. The next state is sampled as $q_{k+1} \sim \delta(q_k, i_k)$. Conversely, the likelihood of a given continuous trace $t = \langle x_0, i_1, x_1, \ldots, i_n, x_n \rangle$ is computed using a sequence of states

q_k with $0 \leq k \leq n$ such that $\forall k \in [0, n] : L(q_k) = A(x_k) \wedge (k \neq n \Rightarrow \delta(q_k, i_{k+1})(q_{k+1}) \neq 0)$. Since determinism is assumed, such a sequence of states is unique if it exists. The likelihood of t can then be calculated from q_k as $L(t \mid \mathcal{M}_c) = \sigma(q_0)(x_0) \prod_{k=1}^{n} \delta(q_{k-1}, i_k)(q_k)\sigma(q_k)(x_k)$.

MDPs and their extension into continuous abstractions are conceptually similar to Moore machines in the sense of von Berg and Aichernig (2025), as the output depends only on the current state. A similar construction for adding continuous data to stochastic Mealy machines is possible. However, there are several subtle differences and discussing them is out of the scope of this paper.

3.2. Using Continuous Data during Automata Learning

In this section, we describe a state-merging algorithm that uses the continuous data in its compatibility check. In particular, our method tests the compatibility of the output-distributions of different states. This is complimentary to established automata learning algorithms, which perform tests on the distributions governing the transition probabilities between states. This allows us to combine our method with established algorithms from the discrete setting, thus considering both aspects of the system as a whole.

Overview. As usual in state-merging, the first step is to construct a PTA from the continuous data with respect to some mapper A. The abstraction is required to avoid the infinite-state issue mentioned in the preliminaries. We also obtain the continuous output map $\hat{\sigma}$ for the PTA, which is used in compatibility checking instead of the abstract values. The compatibility check is based on standard methods from statistical hypothesis testing. Given an FIOA $\mathcal{M} = \langle Q, q_0, I, O, L, \hat{\delta} \rangle$, any partitioning of its states π corresponds to a hypothesis $\mathcal{H}_0$ relating the FIOA to the system $\mathcal{S}$ from which the traces were sampled. In particular, $\mathcal{H}_0$ states that, for each partition $Q_k \in \pi$, all states $q \in Q_k$ correspond to the same state p of the system $\mathcal{S}$. This is in contrast to a more complex hypothesis in which at least one of the partitions Q_k contains a state q that corresponds to a different state p'. If sufficient data is available, $\mathcal{H}_0$ is equivalent to the statement that for each partition $Q_k \in \pi$ all $q \in Q_k$ are compatible in the following sense. First, the unknown distributions $F(q)$ from which the samples $\hat{\sigma}(q)$ are drawn have to be the same distribution $F \in \mathcal{D}(\mathcal{C})$. Formally, this is expressed as $\forall Q_k \in \pi, q_1, q_2 \in Q_k : F(q_1) = F(q_2)$. Second, for all $i \in I$, the samples from the empirical transition function $\hat{\delta}(q, i)$ have to be consistent with respect to how often each representative $o \in O$ is reached. This allows us to decompose $\mathcal{H}_0$ into the two (conjunctive) hypotheses $\mathcal{H}_\sigma$ for the output distributions and $\mathcal{H}_\delta$ for the transition distributions. Established automata learning methods for the discrete setting are commonly based on $\mathcal{H}_\delta$. Since we use our method in conjunction with a discrete learning algorithm, this allows us to focus on $\mathcal{H}_\sigma$, which is unique to the continuous setting. For a specific partition Q_k, the hypothesis $\mathcal{H}_{Q_k}$ whether all samples $\hat{\sigma}(q)$ for $q \in Q_k$ stem from a common distribution can be tested using a k-sample test. Such a test produces a p-value that states how extreme the observations are with respect to $\mathcal{H}_{Q_k}$. The p-values from all partition-input combinations can be combined into a p-value for $\mathcal{H}_\sigma$, which is the conjunction of all $\mathcal{H}_{Q_k}$. The hypothesis $\mathcal{H}_\sigma$ is rejected iff the combined p-value is below a user-specified threshold ϵ. A merge is only accepted if both $\mathcal{H}_\sigma$ and $\mathcal{H}_\delta$ are accepted, where the latter is deferred to the discrete algorithm $\mathcal{A}_D$. If $\mathcal{A}_D$ is also based on p-values, the p-values for $\mathcal{H}_\sigma$ and $\mathcal{H}_\delta$ can be combined into a single p-value, which can not only be used as singular

rejection criterion but also serve as a score for ranking different viable merge candidates. This results in the compatibility check illustrated in Algorithm 1.

Algorithm 1 Compatibility computation

Input: States Q, Inputs I, Partitioning π, Continuous Output Map $\hat{\sigma}$
Parameters: Rejection Threshold ϵ, Discrete Algorithm $\mathcal{A}_D$
Output: Score or Rejection $\perp$

```
begin
    ps ← ∅
    for Q_k ∈ π do
        ps[Q_k] ← k-sample test for ⟨σ̂(q) | q ∈ Q_k⟩
    end
    p ← combine p-values in ps
    result_δ ← A_D(π)
    if p < ε ∨ result_δ = ⊥ then
        return ⊥
    else
        return combine p and result_δ
    end
end
```

Statistical Tests. There are several k-sample tests that can be used in the algorithm. We considered rank tests such as the extension of the Anderson-Darling test by Scholz and Stephens (1987), Kruskal-Wallis tests, van der Waerden scores and normal scores (compare Lehmann and D'Abrera (2006)), as well as tests based on (in-)dependence measures discussed by Panda et al. (2025). Additionally, we also considered the Kolmogorov-Smirnov (KS) test, which is restricted to two samples. Our algorithm can be adapted to the two-sample case as follows. For each partition Q_k and state $q \in Q_k$, we evaluate the KS test for the samples $\hat{\sigma}(q)$ and the joint sample from all $q' \in Q_k$ s.t. $q' \neq q$. This extension of the two-sample KS test into a k-sample test works reasonably well in practice, although it is not canonical to the best of our knowledge.

The independence-based tests are of special interest since they also work for multivariate data, which enables us to handle systems where the output is a vector rather than a single real number. This is in contrast to rank-based tests, which do not generalize well to multivariate data. A naive method for using rank-based tests in the multivariate setting, is to perform the test for each dimension separately and combine the results afterwards. However, this approach does not consider dependencies between variables.

Another parameter that can be adjusted is how the p-values are combined. There is a rich variety of combination methods – compare Nikolitsa et al. (2025) – including methods that can handle weighting such as Stouffer's method and methods based on averaging as discussed by Vovk and Wang (2020).

4. Evaluation

4.1. Setup

We implemented the two methods outlined in the previous section in Python using the state-merging implementation of the automata learning library AALpy. For the statistical tests we used the libraries scipy, hyppo and kSamples by Virtanen et al. (2020), Panda et al. (2020) and Scholz and Zhu (2025) respectively. The evaluation was performed on a CPU cluster with AMD EPYC 9375F 32-Core processors running at 3.85 GHz. The experiments were run in parallel, each using a single core with a one hour timeout.

Evaluation Method. We evaluated our approach ($\mathcal{A}_C$) by comparing it against established state-merging algorithms for MDPs, in particular IOAlergia ($\mathcal{A}_{Al}$) by Mao et al. (2016) and a variant of the likelihood-ratio method ($\mathcal{A}_{LR}$) by Verwer et al. (2010) adapted for the IO case. As a least effort baseline, we also consider the algorithm $\mathcal{A}_U$ computing the unigram model. This model is constructed by introducing a single state for each abstract observation $o \in O$. A transition is added between two states with outputs o_1 and o_2 on input i if $\langle o_1, i, o_2 \rangle$ is a sub-sequence of any trace. Each algorithm is evaluated on a set of benchmark systems using a wide range of parameters. We also study the effect of sampling methods, i.e., the number and length of traces, on the quality of the models. The quality of the resulting models is assessed based on the average step likelihood of data from a testing set that is generated using the same sampling method. For a given benchmark system $\mathcal{S}$ and sampling strategy $\mathcal{G}$, we obtain a training set T_{train} and a test set T_{test} sampled from $\mathcal{S}$ using $\mathcal{G}$. For each algorithm $\mathcal{A}$ in consideration, we learn a continuous abstraction $\mathcal{M}_c = \mathcal{A}(T_{train}, \phi)$ for several parameter combinations ϕ. The parameters include for example the used abstraction A and the rejection threshold ϵ. We then evaluate the geometric mean of the per-step likelihood of the test data on the inferred model, which is given as $\bar{L}_G = \prod_{t \in T_{test}} L(t \,|\, \mathcal{M}_c)^{1/N}$, with $N = \sum_{t \in T_{test}} (|t| + 1)/2$, which provides us with a metric that can be compared across different data generation methods. If even a single step has a likelihood approaching zero, the geometric mean will approach zero as well. While – in general – steps with zero likelihood are indicative of a bad model, they are expected to occur in low data scenarios where the training data do not contain the full extent of the system behavior. This can happen even if a model is input complete, as some abstract output might not have been observed for a specific state-input pair. For this reason, we also report the median per-step likelihood $\tilde{L}_M$. We do not consider the arithmetic mean because it is very sensitive to steps with very high likelihood, which can often dominate the result. This happens for states with narrow distributions regardless of whether these distributions accurately model system behavior or are artifacts due to insufficient data. To sum up, we compare two learned models by checking how well they explain unseen data, measured by the average per-step likelihood of a test dataset that was generated in the same way as the training data.

Except for $\mathcal{A}_U$, each algorithm has some form of confidence parameter ϵ. For each algorithm, this parameter is varied using eight values in a wide range from $0.5 * 10^{-14}$ to 0.1. We refer to a group of configurations that differ only in the ϵ-value used as an ϵ-ensemble. Unless otherwise noted, we aggregate results among ϵ-ensembles and report only the value for the ϵ that achieves the best results with respect to one of the metrics discussed above. Our method has additional parameters such as the statistical compatibility test and

the method for combining p-values. These parameters are compared separately later on. Also, our algorithm needs to be applied in conjunction with an algorithm that accounts for the transition probabilities.

Evaluated Systems. We use several systems to evaluate our approach. This includes three examples from the Gymnasium collection of systems for reinforcement learning: Mountain Car $\mathcal{S}_{MC}$, Cart Pole $\mathcal{S}_{CP}$ and Pendulum $\mathcal{S}_P$. Each of the systems has a multidimensional state space. The (apparent) stochasticity of the systems stems from three factors: (1) each observation is subject to output noise, (2) the initial state is chosen randomly and (3) discretization error. For each system, we consider two cases with one and two observable state variables respectively. For more detail regarding the individual systems we refer to the paper by Towers et al. (2024).

Apart from the Gymnasium environments, we consider three additional systems. The first system considers a robot that is attached on a rail along which it can move. In each step, the robot tries to move a fixed amount to the left or to the right depending on the input of the system, i.e., $|I| = 2$. However, the robot's movement is (1) constrained to the rail, which we set to 16 steps long and (2) imprecise, which is modeled by additive Gaussian noise with a standard deviation of a quarter step. Initially the robot is in the middle of the rail. The output of the system is the position on the rail. We call this system *Deterministic Robot* or $\mathcal{S}_{DR}$ for short.

The second system, *Nondeterministic Robot* ($\mathcal{S}_{NR}$), is almost identical to the first, except that the input to the system determines whether the robot stays in place or moves in a randomly chosen direction. Due to the random choice of the movement direction, the input alone is less indicative of the reached state than for $\mathcal{S}_{DR}$ and assessing observations correctly becomes more important. We expect this to increase the impact of the discretization scheme since it determines the quality of the observations.

The third system is called *Water Tank* ($\mathcal{S}_{WT}$). It consists of a tank with an inlet and a pump which removes water from the tank. The system has two actions: inaction and activating the inlet. Activating the inlet triggers a stochastic process in which the tank is filled. The inflow rate is high initially but stochastically decreases over time until it reaches zero. The pump is turned on if the water level exceeds 80 percent capacity and turned off if the level drops below 20 percent. Additionally, small fluctuations occur. Only the change in water level is observable. The system was designed to be simple yet challenging. In particular, it contains a deterministic but not directly observable component (the pump) and a stochastic but partially controllable component (the inlet).

The sampling strategies have three parameters N_t, n_{min}, n_{max}. They sample N_t sequences from the input alphabet of the system under learning, where each symbol occurs uniformly and the length is chosen according to a uniform distribution over the interval $[n_{min}, n_{max}]$. We consider five different configurations. The first three use short traces with 10 to 20 steps and are denoted as $\mathcal{G}_{s,1}$, $\mathcal{G}_{s,2}$ and $\mathcal{G}_{s,3}$ with 100, 500 and 1000 traces respectively. The data generation method $\mathcal{G}_m$ uses 500 traces of length 40 to 60. Finally, $\mathcal{G}_l$ uses 100 traces of length 100 to 200. We use a simple discretization scheme with N_D evenly sized intervals. The number of discretization steps is given and other parameters are discussed below on a case to case basis.

		$\bar{L}_G$						$\bar{L}_M$					
	T/O	$\mathcal{A}_U$	$\mathcal{A}_{Al}$	$\mathcal{A}_{LR}$	$\mathcal{A}_C^{DC}$	$\mathcal{A}_C^{KS}$	$\mathcal{A}_C^{M}$	$\mathcal{A}_U$	$\mathcal{A}_{Al}$	$\mathcal{A}_{LR}$	$\mathcal{A}_C^{DC}$	$\mathcal{A}_C^{KS}$	$\mathcal{A}_C^{M}$
$\mathcal{A}_U$	0.0	-	70.0	71.4	**85.7**	84.2	84.4	-	78.3	87.5	97.1	**97.3**	89.6
$\mathcal{A}_{Al}$	0.0	0.0	-	57.1	**85.7**	78.9	81.0	0.0	-	71.4	**97.1**	94.7	87.9
$\mathcal{A}_{LR}$	6.7	19.6	33.9	-	**85.7**	78.9	66.0	10.7	26.7	-	**97.1**	89.4	60.7
$\mathcal{A}_C^{DC}$	42.3	8.5	8.5	5.7	-	**44.4**	11.4	2.8	2.8	2.8	-	**48.1**	14.2
$\mathcal{A}_C^{KS}$	37.2	10.5	15.7	10.5	**48.1**	-	18.4	2.6	5.2	10.5	**48.1**	-	18.4
$\mathcal{A}_C^{M}$	3.3	10.3	13.7	26.7	**82.8**	73.6	-	10.3	12.0	39.2	**85.7**	81.5	-

Table 1: Aggregated results for the *core* benchmark including the timeout ratio T/O and how often the column algorithm outperformed the row algorithm.

4.2. Results

We initially performed a screening process to narrow down the parameter space. This initial test phase showed that $\mathcal{A}_{LR}$ usually produces better models than $\mathcal{A}_{Al}$ with respect to our metrics. For this reason, we chose $\mathcal{A}_{LR}$ as the algorithm responsible for testing the transition distribution used in $\mathcal{A}_C$, i.e., $\mathcal{A}_D = \mathcal{A}_{LR}$ in our evaluation. Furthermore, from the statistical tests provided by kSamples, hyppo and scipy, we eliminated all but three tests: the rank-based KS test from scipy, the independence-based Distance Correlation (DC) test from hyppo and Multivariate Analysis of Variance (MANOVA or simply M) also provided by hyppo. The other tests were not able to consistently outperform those three tests in terms of quality or required time.

Core Bechmark. The core benchmark set consists of the systems with univariate observations. It considers all five data sampling strategies. We use two different discretization schemes with $N_D \in \{4, 8\}$ steps. We evaluated $\mathcal{A}_U$, $\mathcal{A}_{Al}$, $\mathcal{A}_{LR}$ as well as our approach $\mathcal{A}_C$ with the three hypothesis tests KS, DC and MANOVA. The p-values are combined using Fisher's method. A qualitative evaluation of the results is given in Table 1. The first column shows the percentage of how many ϵ-ensembles failed to compute even a single model in time. These ϵ-ensembles are not considered in the following. Each of the remaining columns shows the percentage of how often the respective algorithm outperformed the other algorithms specified by the rows with respect to the two metrics $\bar{L}_G$ and $\bar{L}_M$. Both metrics show the same trends, although more prominently for the median. The less extreme values for the geometric mean are due to occasional outliers where $\bar{L}_G = 0$. While DC and KS both exhibit a high number of timeouts, the models they produce if they finish are usually of higher quality than those of the other algorithms. MANOVA strikes a balance of quality and time requirements. While it still outperforms $\mathcal{A}_{LR}$ more often than not, it can not compete with KS and DC in terms of model quality. However, MANOVA is fast to compute and results in even less timeouts than $\mathcal{A}_{LR}$. This might seem surprising at first glance, since it uses $\mathcal{A}_{LR}$ internally. However, incorrectly merging two states can induce additional structure in the state space of the learned model and thus artificially inflate it, leading to an increase of options in merging. Therefore spending more time on individual compatibility checks can decrease overall runtime if it increases the quality of merges.

To evaluate how much our method can improve the quality of the learned models, we consider specific data generation and discretization parameters. In particular, Table 2 shows

		$\mathcal{S}_{DR}$		$\mathcal{S}_{NR}$		$\mathcal{S}_{WT}$		$\mathcal{S}_{MC}$		$\mathcal{S}_{CP}$		$\mathcal{S}_{P}$	
	Alg	$\bar{L}_G$	t	$\bar{L}_G$	t	$\bar{L}_G$	t	$\bar{L}_G$	t	$\bar{L}_G$	t	$\bar{L}_G$	t
$D_N = 4$	$\mathcal{A}_U$	0.149	< 1	0.165	< 1	1.06	< 1	2.56	< 1	7.48	< 1	0.109	< 1
	$\mathcal{A}_{Al}$	0.325	< 1	0.188	4	1.06	< 1	2.77	~ 1	9.29	< 1	0.110	3
	$\mathcal{A}_{LR}$	0.361	4	0.219	4	1.27	19	2.56	< 1	9.39	14	0.113	52
	$\mathcal{A}_C^{DC}$	-	-	0.263	2862	1.42	1956	-	-	0.00	2867	**0.135**	2362
	$\mathcal{A}_C^{KS}$	**0.473**	936	**0.264**	669	**1.43**	497	-	-	**9.84**	565	0.122	2048
	$\mathcal{A}_C^{M}$	0.392	188	0.189	13	1.24	42	**3.47**	326	9.67	80	0.123	566
$D_N = 8$	$\mathcal{A}_U$	0.239	< 1	0.237	< 1	**1.61**	< 1	3.47	< 1	8.97	< 1	0.139	< 1
	$\mathcal{A}_{Al}$	0.416	~ 1	0.248	< 1	**1.61**	< 1	4.03	< 1	10.21	< 1	0.139	< 1
	$\mathcal{A}_{LR}$	0.488	14	0.268	34	0.00	138	4.28	33	9.59	10	0.097	859
	$\mathcal{A}_C^{DC}$	-	-	0.319	2000	-	-	-	-	**10.36**	726	**0.155**	2105
	$\mathcal{A}_C^{KS}$	**0.574**	997	**0.321**	936	0.00	2147	-	-	9.40	1030	-	-
	$\mathcal{A}_C^{M}$	0.509	211	0.286	160	1.50	235	**4.41**	290	10.23	54	0.144	807

Table 2: Per-step likelihood $\bar{L}_G$ and execution time t for data sampled using $\mathcal{G}_{s,2}$.

the best results obtained in terms of $\bar{L}_G$ for $\mathcal{G}_{s,2}$ with $N_D \in \{4, 8\}$. This data generation was chosen for the fact that it produces more data than $\mathcal{G}_{s,1}$ but in contrast to $\mathcal{G}_{s,3}$, $\mathcal{G}_m$ and $\mathcal{G}_l$, allows most algorithms to complete in time. For each system $\mathcal{S}$, the table shows the value of $\bar{L}_G$ as well as the time required to compute the result. While KS and DC are slower than the other algorithms, they also produce significantly better results. In particular, an improvement of about 20 percent compared to the best non-$\mathcal{A}_C$ method is observed several times, with a peak of over 30 percent. For the two cases where KS / DC result in $\bar{L}_G = 0$, the respective $\bar{L}_M$ values are still higher than any of the other algorithms. The highest improvement across all experiments was measured as 96.7 % for $\mathcal{S}_{CP}$ with $\mathcal{G}_{s,1}$ and $N_D = 4$. The worst degradation on performance with respect to non-$\mathcal{A}_C$ algorithms on the other hand was -8.2 % for $\mathcal{S}_{WT}$ with $\mathcal{G}_l$ and $N_D = 8$. These results clearly show, that considering continuous data during the state-merging process is beneficial for quality of the resulting models.

Multivariate Systems. Next we handle systems with multivariate output. For this, we slightly adapt the Gymnasium-based systems by extending the observable part of the state space of each system by a second variable. The discretization is applied to each variable separately, resulting in N_D^2 potential outputs. For this reason, we consider $N_D \in \{2, 4, 8\}$. The additional value 2 leads to $|O| = 4$, similar to $N_D = 4$ in the core benchmark. The benchmark set is similar to the core set otherwise. A quantitative analysis similar to Table 1 is provided in Table 3. While the table exhibits some of the trends of the core set, they are not as clear as previously. A major difference is that, in terms of $\bar{L}_G$ LR and KS perform significantly worse than before. Some of that is explained by the fact that both tend to learn bigger models compared to the other algorithms with an average number of states around 45 compared to 30. This means that the data is spread out among the automaton states which makes issues related to input completeness and missing outputs more common, which implicitly penalizes large models. This holds particularly for low data scenarios, which is especially detrimental for KS, as it tends to time out with too much data. The issue is exacerbated by the high number of discretized values in this benchmark. On the other hand, a high number of discretized values strengthens the unigram model and algorithms

		$\bar{L}_G$						$\bar{L}_M$					
	T/O	$\mathcal{A}_U$	$\mathcal{A}_{Al}$	$\mathcal{A}_{LR}$	$\mathcal{A}_C^{DC}$	$\mathcal{A}_C^{KS}$	$\mathcal{A}_C^{M}$	$\mathcal{A}_U$	$\mathcal{A}_{Al}$	$\mathcal{A}_{LR}$	$\mathcal{A}_C^{DC}$	$\mathcal{A}_C^{KS}$	$\mathcal{A}_C^{M}$
$\mathcal{A}_U$	0.0	-	**62.2**	42.9	56.1	38.5	56.4	-	82.2	**92.9**	92.7	69.2	74.6
$\mathcal{A}_{Al}$	0.0	0.0	-	38.1	45.2	23.5	**47.5**	0.0	-	81.0	**96.8**	58.8	85.0
$\mathcal{A}_{LR}$	7.5	26.2	33.3	-	**43.3**	23.5	38.5	4.76	16.7	-	**76.7**	52.9	51.3
$\mathcal{A}_C^{DC}$	35.2	22.0	29.0	20.0	-	**33.3**	22.0	4.88	0.0	20.0	-	**54.2**	12.2
$\mathcal{A}_C^{KS}$	63.0	46.2	**58.8**	47.1	41.7	-	46.2	30.8	41.2	**47.1**	45.8	-	38.5
$\mathcal{A}_C^{M}$	9.3	14.6	30.0	33.3	**53.7**	38.5	-	18.2	15.0	48.7	**87.8**	61.5	-

Table 3: Aggregated results for the *multivariate* benchmark

that tend toward it such as $\mathcal{A}_{Al}$, which further closes the gap. Still KS is a strong contender that outperforms the other algorithms more often than not when considering $\bar{L}_M$.

Maximal Abstraction. In the most extreme case, the abstraction maps each concrete value to the same representative, i.e. $|O| = 1$. We refer to this as *trivial abstraction* $A_\top$. This choice of abstraction, while nonsensical in most cases, exemplifies the strength of the proposed method. Under this abstraction, each algorithm based on the abstract values alone degenerates to the unigram algorithm $\mathcal{A}_U$. In contrast, our method is still capable of distinguishing states based on the continuous data associated with each state. Averaged over the core benchmark, the best performing model learned with our algorithm under $A_\top$ has a 52.3 % higher per-step likelihood than the corresponding unigram model. This is most extreme for systems where the system state is fully observable such as $\mathcal{S}_{DR}$, where $\bar{L}_G$ is up to 4.3 time higher than for the unigram model. These models can even be competitive with models learned from more data and with a finer abstraction. For example, the model inferred by $\mathcal{A}_C^{KS}$ from the multivariate version of $\mathcal{S}_{MC}$ using $\mathcal{G}_{s,1}$ under $A_\top$ outperforms all non-$\mathcal{A}_C$ models not using $\mathcal{G}_{s,3}$ or $\mathcal{G}_{s,2}$ with $N_D = 8$ with respect to both $\bar{L}_G$ and $\bar{L}_M$. That is to say that it outperforms models learned from five times as much data with sixteen output labels. A similar but less pronounced effect occurs for the multivariate $\mathcal{S}_{CP}$.

Combining P-Values. To investigate the effect of the method for combining p-values, we considered all univariate systems with $\mathcal{G}_{s,2}$ and $N_D = 4$. In this paper, we consider the Cauchy Combination Test (CCT), the tests by Fisher (F) and Stouffer (S) as well as the geometric mean (G). The latter two support weighting, in which case we consider the unweighted case and weighting with the sample size (W). Weighting is slightly adjusted in the two-sample case of KS. For each test involving a sample from a single node and the rest of the partition, we consider the sample size of the single node rather than the combined sample. For the DC and KS hypothesis test, the results of these experiments clearly suggest using Fisher's method or the geometric mean with weighting, with a bias towards F for KS and G+W for DC. While the time requirements are roughly similar for F and G+W, three ϵ-ensembles timed out for F, which favors G+W. In the case of MANOVA, Fisher's method still outperforms each other method at least fifty percent of the time, although the situation is much less clear than for the other algorithms. For the exact results, we refer to Table 4 in Appendix A. A more thorough evaluation of this aspect would be desirable but is out of scope for this paper.

5. Conclusion

Summary. In this paper, we discussed the problem of inferring approximate automaton models from reactive systems with continuous outputs with stochastic behavior. We proposed a simple method for adding continuous output distribution to existing MDP models in a post-processing step. This process results in a new type of model, which we called a continuous abstraction. The MDP model required for this approach can be inferred using existing automata learning methods in conjunction with a mapper that discretizes the continuous observations. We then provided a method that takes the continuous data into account during inference. The algorithm uses non-parametric k-sample tests to determine whether a potential merge of two nodes should be rejected. The method was evaluated in comparison to existing automata learning methods relying solely on the discretized data. The evaluation was carried out using both established examples from the field of reinforcement learning as well as handcrafted systems that exhibit interesting characteristics.

Related Work. Only little work considers the question of inferring automaton models from reactive systems with continuous outputs that exhibit stochastic behavior. Nevertheless, many contributions exist for either field in isolation. An interesting approach not directly related to automata is provided by Hazan et al. (2017). They propose a method that allows inferring Linear Dynamical Systems from data. In this formalism, the state of the system and its output are governed by a linear equation system subject to noise. An overview on inference of probabilistic automata is given by Verwer and Hammerschmidt (2025). Pellegrino et al. (2016) provide an inference algorithm for an automata formalism with continuous outputs. However their method produces deterministic models where states statically approximate a single fixed output rather than fitting a distribution. Extended Finite State Machines (EFSMs) are a very expressive but fully deterministic formalism where states are augmented by data variables. It considers inputs with parameters. The input parameters and the internal state are used in guards to determine which transition to take and also govern how the internal state is updated and which output is emitted. Inference techniques for this formalism have been developed by Mariani et al. (2017), Foster et al. (2021) and Chivilikhin et al. (2019) with varying degrees of completeness. While theoretically suitable for continuous behavior, EFSMs are mainly applied in discrete settings. This is in contrast to hybrid automata, which is a formalism explicitly designed to describe hybrid systems. In this formalism, each state is associated with a system of differential equations that govern the trajectories of internal variables. Gurung et al. (2023) and Poli et al. (2021) discuss inference algorithms for this formalism in the case of deterministic and stochastic systems respectively. The latter approach targets a similar type of system as our method. However, it is conceptually very different and does not directly translate to our setting. For example, they assume that the behavior is fully determined by an observable mode.

Discussion. The evaluation given in Section 4 supports the following key findings of this paper. (1) While the formalism of continuous abstractions may not be a class of automata that accurately describes an interesting class of reactive systems, it still is a useful tool in the approximation of such systems. In particular, we showed that considering the continuous values during model inference is a valuable idea that can significantly increase the quality of the resulting models. (2) For the proposed algorithm, a trade-off has to

be made between execution time and model quality. If quality is the main priority, our algorithm should be instantiated using either Kolmogorov-Smirnov or Distance Correlation as the k-sample test. In this case, either Fisher's test or a weighted geometric mean should be used for combining p-values. If execution time is an issue, MANOVA can be used as a k-sample test to obtain a model in less time.

In the future, we would like to use this approach to learn environment models for reinforcement learning algorithms. This is useful for both training and verifying reinforcement learning agents as discussed, e.g., by Muskardin et al. (2023) and Tappler et al. (2024) respectively.

Acknowledgements. The work in this paper was supported by the Austrian Science Fund (FWF) Cluster of Excellence Bilateral AI under contract number 10.55776/COE12.

References

Fides Aarts, Bengt Jonsson, Johan Uijen, and Frits W. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *Formal Methods Syst. Des.*, 46(1):1–41, 2015. doi: 10.1007/S10703-014-0216-X. URL https://doi.org/10.1007/s10703-014-0216-x.

Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *Formal Aspects Comput.*, 28(2):233–263, 2016. doi: 10.1007/s00165-016-0355-5. URL https://doi.org/10.1007/s00165-016-0355-5.

Daniil Chivilikhin, Vladimir Ulyantsev, Anatoly Shalyto, and Valeriy Vyatkin. Function block finite-state model identification using SAT and CSP solvers. *IEEE Trans. Ind. Informatics*, 15(8):4558–4568, 2019. doi: 10.1109/TII.2019.2891614. URL https://doi.org/10.1109/TII.2019.2891614.

Michael Foster, John Derrick, and Neil Walkinshaw. Reverse-engineering EFSMs with data dependencies. In David Clark, Héctor D. Menéndez, and Ana Rosa Cavalli, editors, *Testing Software and Systems - 33rd IFIP WG 6.1 International Conference, ICTSS 2021, London, UK, November 10-12, 2021, Proceedings*, volume 13045 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2021. doi: 10.1007/978-3-031-04673-5_3. URL https://doi.org/10.1007/978-3-031-04673-5_3.

Artur Gramacki. *Nonparametric kernel density estimation and its computational aspects*, volume 37. Springer, 2018.

Amit Gurung, Masaki Waga, and Kohei Suenaga. Learning nonlinear hybrid automata from input-output time-series data. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I*, volume 14215 of *Lecture Notes in Computer Science*, pages 33–52. Springer, 2023. doi: 10.1007/978-3-031-45329-8_2. URL https://doi.org/10.1007/978-3-031-45329-8_2.

Elad Hazan, Karan Singh, and Cyril Zhang. Learning linear dynamical systems via spectral filtering. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M.

Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6702–6712, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/165a59f7cf3b5c4396ba65953d679f17-Abstract.html.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In Vasant G. Honavar and Giora Slutzki, editors, *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998. doi: 10.1007/BFB0054059.

Erich Leo Lehmann and Howard JM D'Abrera. *Nonparametrics: statistical methods based on ranks*, volume 464. Springer New York, 2006.

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

Leonardo Mariani, Mauro Pezzè, and Mauro Santoro. GK-Tail+ an efficient approach to learn software models. *IEEE Trans. Software Eng.*, 43(8):715–738, 2017. doi: 10.1109/TSE.2016.2623623. URL https://doi.org/10.1109/TSE.2016.2623623.

Edi Muskardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. AALpy: An active automata learning library. *Innov. Syst. Softw. Eng.*, 18(3):417–426, 2022. doi: 10.1007/S11334-022-00449-3. URL https://doi.org/10.1007/s11334-022-00449-3.

Edi Muskardin, Martin Tappler, Bernhard K. Aichernig, and Ingo Pill. Reinforcement learning under partial observability guided by learned environment models. In Paula Herber and Anton Wijs, editors, *iFM 2023 - 18th International Conference, iFM 2023, Leiden, The Netherlands, November 13-15, 2023, Proceedings*, volume 14300 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2023. doi: 10.1007/978-3-031-47705-8_14. URL https://doi.org/10.1007/978-3-031-47705-8_14.

Evgenia K Nikolitsa, Panagiota I Kontou, and Pantelis G Bagos. metacp: A versatile software package for combining dependent or independent p-values. *BMC bioinformatics*, 26(1):109, 2025.

Sambit Panda, Satish Palaniappan, Junhao Xiong, Eric W. Bridgeford, Ronak Mehta, Cencheng Shen, and Joshua T. Vogelstein. hyppo: A comprehensive multivariate hypothesis testing python package, 2020.

Sambit Panda, Cencheng Shen, Ronan Perry, Jelle Zorn, Antoine Lutz, Carey E Priebe, and Joshua T Vogelstein. Universally consistent k-sample tests via dependence measures. *Statistics & Probability Letters*, 216:110278, 2025.

Gaetano Pellegrino, Christian A. Hammerschmidt, Qin Lin, and Sicco Verwer. Learning deterministic finite automata from infinite alphabets. In Sicco Verwer, Menno van Zaanen,

and Rick Smetsers, editors, *Proceedings of the 13th International Conference on Grammatical Inference, ICGI 2016, Delft, The Netherlands, October 5-7, 2016*, volume 57 of *JMLR Workshop and Conference Proceedings*, pages 120–131. JMLR.org, 2016. URL http://proceedings.mlr.press/v57/pellegrino16.html.

Andrea Pferscher and Bernhard K. Aichernig. Learning abstracted non-deterministic finite state machines. In Valentina Casola, Alessandra De Benedictis, and Massimiliano Rak, editors, *Testing Software and Systems - 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9-11, 2020, Proceedings*, volume 12543 of *Lecture Notes in Computer Science*, pages 52–69. Springer, 2020. doi: 10.1007/978-3-030-64881-7_4. URL https://doi.org/10.1007/978-3-030-64881-7_4.

Michael Poli, Stefano Massaroli, Luca Scimeca, Sanghyuk Chun, Seong Joon Oh, Atsushi Yamashita, Hajime Asama, Jinkyoo Park, and Animesh Garg. Neural hybrid automata: Learning dynamics with multiple modes and stochastic transitions. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 9977–9989, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5291822d0636dc429e80e953c58b6a76-Abstract.html.

Fritz Scholz and Angie Zhu. *kSamples: K-Sample Rank Tests and their Combinations*, 2025. URL https://CRAN.R-project.org/package=kSamples. R package version 1.2-12.

Fritz W Scholz and Michael A Stephens. K-sample anderson–darling tests. *Journal of the American Statistical Association*, 82(399):918–924, 1987.

Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L^*-based learning of markov decision processes (extended version). *Formal Aspects Comput.*, 33(4-5):575–615, 2021. doi: 10.1007/S00165-021-00536-5. URL https://doi.org/10.1007/s00165-021-00536-5.

Martin Tappler, Edi Muskardin, Bernhard K. Aichernig, and Bettina Könighofer. Learning environment models with continuous stochastic dynamics - with an application to deep RL testing. In *IEEE Conference on Software Testing, Verification and Validation, ICST 2024, Toronto, ON, Canada, May 27-31, 2024*, pages 197–208. IEEE, 2024. doi: 10.1109/ICST60714.2024.00026. URL https://doi.org/10.1109/ICST60714.2024.00026.

Mark Towers, Ariel Kwiatkowski, Jordan K. Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *CoRR*, abs/2407.17032, 2024. doi: 10.48550/ARXIV.2407.17032. URL https://doi.org/10.48550/arXiv.2407.17032.

Sicco Verwer and Christian A. Hammerschmidt. Flexfringe: Modeling software behavior by learning probabilistic automata. *Log. Methods Comput. Sci.*, 21(3), 2025. doi: 10.46298/LMCS-21(3:31)2025. URL https://doi.org/10.46298/lmcs-21(3:31)2025.

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2010. doi: 10.1007/978-3-642-15488-1_17. URL https://doi.org/10.1007/978-3-642-15488-1_17.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Benjamin von Berg and Bernhard K. Aichernig. Extending AALpy with passive learning: A generalized state-merging approach. In Ruzica Piskac and Zvonimir Rakamaric, editors, *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part IV*, volume 15934 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 2025. doi: 10.1007/978-3-031-98685-7_6. URL https://doi.org/10.1007/978-3-031-98685-7_6.

Vladimir Vovk and Ruodu Wang. Combining p-values via averaging. *Biometrika*, 107(4): 791–808, 2020.

Appendix A. Additional Tables

	PCM	$\mathcal{S}_{DR}$	$\mathcal{S}_{NR}$	$\mathcal{S}_{WT}$	$\mathcal{S}_{MC}$	$\mathcal{S}_{CP}$	$\mathcal{S}_{P}$
DC	CCT	0.443	0.250	1.283	**3.677**	10.103	0.123
	F	-	0.263	1.426	-	0.000	0.135
	S	0.380	0.221	1.275	3.593	9.926	0.120
	S+W	0.460	0.219	1.307	3.484	10.060	0.119
	G	0.459	0.262	1.399	3.035	**10.592**	**0.136**
	G+W	**0.476**	**0.266**	**1.489**	3.318	10.191	**0.136**
KS	CCT	0.340	0.201	1.174	2.610	9.600	0.111
	F	**0.473**	**0.264**	**1.431**	-	9.835	**0.122**
	S	0.346	0.205	1.198	2.598	9.094	0.112
	S+W	0.353	0.194	1.300	2.605	9.291	0.110
	G	0.398	0.228	0.000	**3.767**	8.766	0.117
	G+W	0.462	0.258	1.410	3.673	**9.884**	0.120
M	CCT	**0.399**	0.196	1.162	3.415	9.746	0.118
	F	0.392	0.189	**1.248**	**3.466**	9.673	**0.123**
	S	0.378	0.198	1.192	3.370	9.832	0.118
	S+W	0.377	**0.200**	1.132	3.408	**10.234**	0.120
	G	0.374	0.187	1.141	3.298	9.315	0.119
	G+W	0.375	0.192	1.089	3.295	9.181	0.120

Table 4: Average per-step likelihood for different p-value combination methods.