A quick dive into concept learning

Léo Henry

November 30, 2018

This document presents the most interesting parts of my readings on concept learning and mining. A concept is in this case a classification of instances between the one that satisfy its rules (and instantiate the concept) and the one that do not. I focused mainly on learning methods using finite state machine as models, and oriented my reading towards the addition of explicit time.

Concept learning seems to motivates works from very different communities, so instead of trying to organize all of it as a coherent whole, I propose a list of approaches, trying to give insights and group works based on different features. This is then completed by a list of articles.

1 Reading Keys

In this section, the papers are organized using 4 lecture keys. First, different applications are presented. Next, the algorithms are listed, and some emphasis is given to the theoretical articles and their connections. Eventually, the different kinds of models are mentioned and compared.

1.1 Applications

1.1.1 Cyber-Physical Production Systems

A cyber-physical production system (CPPS) is a usual component of industrial production chains. Being able to efficiently test it, and to detect faults and failures is hence of the utmost importance for both economical and security reasons. For this reason, some work as been done in the learning of their specifications [Mai14; MNE15; NL15; WLN17; MNJ+11; TDH00; CO94]. CPPSs testing has in general been a motivation for the development of algorithms learning automata from *positive examples only*. Indeed, acquiring negative examples is often simply unfeasible or incredibly costly (a word out of the language is an unfeasible execution of the system). While it has been long proved that negative examples are necessary to infer the class of regular languages [Gol67], it is also known that some languages can be learned with probability one [Ang88]. Since this result, researchers have tried to obtain an efficient algorithm to learn probabilistic automata from positive examples only. One of the best known algorithms is ALERGIA [CO94]. The idea of this algorithm is first to compute a prefix tree acceptor representing every available execution, and then to merge some states in a top-down manner (from the root

to the leafs). Nodes are merged when the sub-tree rooted in them have a similar language with high probability.

The main limitation of this algorithm is to check only local information in order to perform the merges. This forbids to obtain a bound on the divergence between the language depicted in the samples and the obtained one. The algorithm MDI (Minimal Divergence Inference) [TDH00] proposes to correct this by computing the Kullback-Leibler divergence at each step. The criterion to merge nodes becomes 'is the reduction of the model interesting enough to suffer the divergence'. It hence tries to balance between model complexity and entropy, or generalization and precision.

So far, the model learned were probabilistic (deterministic) automata. But as often with industrial processes, CPPSs have important real-time constraints. For this reason, researchers have tried to explicitly include time in their models. One of the early attempts is the BUTLA (Bottom Up Timing Learning Algorithm) [MNJ+11], that learns a probabilistic timed automaton. The merging strategy is bottom-up to avoid the recursive compatibility checks in the subtrees. The introduction of time motivates the apparition of a new operator: the split operator, that separates a transition in disjoints timed domains. The authors rely on domain specific knowledge to advocate a split performed to separate modes in the probability distribution of a transition rather than a difference in the sub-automata. This idea is re-exploited in [MNE15] to propose a preprocessing of the data to separate the different modes, suppressing the splitting operation.

All the aforementioned methods are *offline*. Yet, the modern CPPSs can generate a great number of data in a very short time, hence motivating the investigation of *online* learning algorithms [Mai14]. This forces to adapt the previous techniques, as they all rely on the construction of the prefix tree acceptor, representing every available samples. OTALA [Mai14] is an online adaptation of BUTLA, allowing online passive learning from positive examples only in an incremental way. It learns a deterministic real time automaton (only one clock, reset at every transition). Notably, it requires a stopping criterion if the data can be provided in an endless stream. OTALA has next been extended to a parallel framework [WLN17] by trying to measure the correlation between events in different components.

[NL15] is to my knowledge the most recent overview of the learning methods for CPPSs. It identifies 3 key-points for the domain and deduce a research agenda from them:

- A general *learnable* model is needed for CPPSs,
- this model should deal with timed and hybrid constraints,
- more work should be done at the level of the component, instead of always focusing on root causes.

Research leads:

- ◇ Modeling should be data-driven (requires a joint work from learning and diagnosis communities)
- ◇ Hybrid models with an explicit modeling of time should be investigated
- ◇ The needs of experts should be taken into account, providing symptoms of the problems atop of the root cause explanation.

1.1.2 Software Analysis and Test

Software analysis has for long been an important application of formal methods. In the case of model learning, the specificity of software is that it allows to leverage some black-box hypotheses by making assumption on the structure of the observed object. One interesting point is that this specificity can be used to learn more complex models (such as automata with multiple clocks). There are two main models that can be learned out of a software

Specification A specification describes what is possible with a given piece of code. It is supposed to describe every possible behaviors, and probably some of the failing executions.

Normal use A behavior model of normal use describes how a piece of code (say an object, or a class) is indeed used. It does not aims completeness, but instead focuses on what the user indeed does.

These models can be used for a broad range of software engineering tasks, such as generating test cases, identifying anomalies, detecting inefficiencies or helping debugging. Different approaches have hence collaborated to obtain formal models of software in a formal way [PMM17; SNF17; DKM+12; DKM+10; LMP08; BF72; CW98; WML02]. As for CPPSs, learning is mainly done from positive examples, although exceptions and errors are now considered as part of these positive examples by some papers, making the approach a bit broader.

One of the earliest works underlying learning for software is the k-Tail algorithm, first introduced in the 70s from a very theoretic point of view [BF72] as a variation on the work of Nerode [Ner58]. The algorithm was then reformulated in [CW98] and compared with two other approaches (based on neural networks and Markov chains) for the learning of software models. This algorithm proposes to synthesize a model "locally", by focusing on subsets of their behaviors. More precisely, the algorithms compares nodes in two traces by looking at their "k-future", the k next nodes in this execution. If they are equals, then the nodes are merged. This approach was then extended to models with parameters [LMP08] with the gk-Tail algorithm and to timed models, with the Timed k-Tail algorithm [PMM17]. These two algorithm propose to construct intervals between the values of parameters - clocks that are merged. In the case of Timed k-Tail, it is interesting to notice that this approach allows to learn a model with multiple clocks (while still minimizing their number during the construction), mostly designed to target nested behaviors. It is interesting to notice that the authors propose two different ways to approximate the guards: by constructing a probability distribution and selecting an interval with a high probability to include the possible duration of an execution, or simply by enlarging the interval by a factor ϵ.

An other interesting approach is proposed in [DKM+10] where the authors propose to combine test and learning. A first model of the system is generated from the result of a test suite, and then mutations of the test suite are generated (such that all methods are called in all states). This allows to enrich the model. This work was then extended to avoid the need for an initial test suite and generate different types of new tests [DKM+12]. This extension allows to iterate on the learning / testing loop and generate new tests. The comparison of these methods is performed in the latter article. As they use a coverage

criterion to direct their tests, the methodology of this work is of great interest to interleave test and learning even in other frameworks. This approach is implemented in a tool named TAUTOKO.

Other approaches exists, closer to the abstract interpretation. We will not detail them here, but two pointers are worth mentioning. A tool named TREM implements a set of methods to learn regular expressions as specifications [SNF17]. The user can propose a general scheme, which is then instantiated automatically. This is especially useful to learn invariants, that are under the focus of a part of the software learning community. From an other point of view, some approaches propose to combine static analysis and dynamic learning of finite states machines to learn an API for high level languages [WML02] (their applications are on JAVA). The authors highlight that both approaches can obtain different insights on the behavior of a piece of code.

1.1.3 General Specification Mining

Some works are directly focused on specification mining without targeting any given application. As motivated by the applications, as the one already presented, most specification mining articles consider passive learning tasks from positive examples. The ALERGIA [CO94] and MDI [TDH00] algorithms were of those, even if they were presented earlier to ground the latter algorithms more focused on CPPSs. Same goes for the Nerode [Ner58] and k-Tail [BF72] algorithms in their first versions on the software side. Other works exists. [VWW08] describes the general scheme of offline learning of deterministic real-time automata that is shared by a lot of other works. They propose to use statistical tests to decide on merges and splits (specifically they reuse the already used Chi-square test and add a Kolmogorov-Smirnov test to further exploit the timed information).

The greater distance with the applications pushes these more general articles to get rid of the implementations in favor of more theoretical analysis. For example, [CT04] proposes a state merging method based on the Kullback-Leibler divergence to learn a probabilistic determinist automata, similar as MDI, but review the positive and negative complexity result and proves with their algorithm that these automata are PAC-learnable (Probably Approximately Correct learning). We reproduce the definition for KL divergence of [CT04]:

Definition 1.1: PAC learnability

Given a class of stochastic languages or distributions C over Σ^* , an algorithm A KL-Probably Approximately Correctly (KL-PAC)-learns C if there is a polynomial q such that for all c in C , all $\epsilon > 0$ and $\delta > 0$, A is given a sample S_m and produces a hypothesis H, such that $Pr[D(c||H) > \epsilon] < \delta$ whenever $m > q(1/\epsilon, 1/\delta, |c|)$, where $|c|$ is some measure of the complexity of the target and $D(c||H) = \Sigma_{s \in S_m} c(s) log(c(s)/H(s))$ is the KL-divergence[a], with running time bounded by a polynomial in m plus the total length of the strings in S_m .

[a] $c(s)$ stands for the probability of the word s in the language of c

1.1.4 Active Learning

The active learning framework differs greatly from specification mining. It was introduced by D. Angluin [Ang87] to leverage the restrictions found on learning from random given samples by Gold [Gol67; Gol78]. The author argues that in a human learning process you can assume the learner to be "helpful", and interacts with the learner. She takes the example of a human specialist trying to train an expert system. To quantify this helpfulness would not make a lot of sense, but Angluin proposes a "minimally adequate teacher" (MAT) that would be able to answer two types of questions

- Membership queries, where the learner proposes a word. The teacher then replies yes or no depending on the word being part of the language to learn.
- Equivalence queries correspond to conjectures from the learner, who wants to know whether or not the model it has learned yields the correct language. The teacher answers either yes, or provide a counterexample to direct the learner.

This teacher allows Angluin to define the L^* algorithm, that can learn regular languages. This algorithm identifies nodes by using Nerode's congruence. Established as a proof of concept it was afterward refined to obtain a better data structure [RS89; KV94].

The principal difficulties to obtain a teacher are come from the equivalence queries. Indeed this requires the teacher to dispose of a model (presumed perfect) of the targeted language, and to be able to communicate with the learner not only with examples, but with models, and hence to share a modelling language with it. The author proposes to use a criterion of approximate identification to replace the ability to answer equivalence queries by a random sampling oracle.

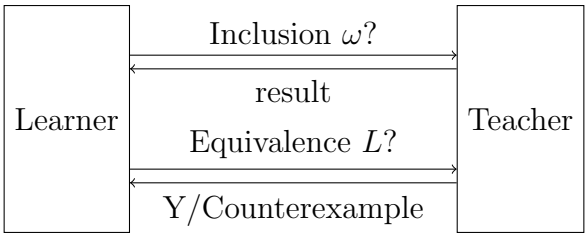

Figure 1: The active learning framework

Extending the MAT framework, an algorithm to learn a black box input-output system with active learning has recently been proposed [JV17]. The model used is the one of mealy machines with timers (with some restrictions on determinism and null delays). The answers to membership queries correspond to words that are produced from the provided inputs instead of yes/no answers. The learning uses an equivalence between a timed and an untimed semantic of mealy machines. This allows to use an untimed learner on a timed teacher, by the mean of an adapter module. A group of researchers centered around the Radboud university of Nijmegen (including F. Vaandrager) are working on the integration of model-based testing in (active) learning methods[1], arguing that tests and learning are complementary tools.

[1] http://www.sws.cs.ru.nl/Teaching/LearningAndTesting

Olga Grinchtein's thesis [Gri08] proposes an other approach to the active learning of timed systems, this times by focusing on event recording automata (ERAs). The highlighted interest of this class of TAs is that the set of clocks and their resets are known. The author proposes algorithms for two kind of models: deterministic ERA (DERA) and event deterministic ERA (EDERA), that only allow 1 outgoing transition with a given letter in a given locality.

1.1.5 Miscellaneous

The formal approach to learning is applied in a broad variety of fields, and some (very partial) pointers are mentioned here. Some researchers have tried to use timed automata (using UPPAAL) to model "third generation neurons" [CDD17] targeted to neuro-science studies. Their main interest was to use the formal models to reproduce behaviors observed in real neurons, and then connect them to be able to share their impulses. The interest of the TAs is their ability to accumulate inputs over-time (in a discrete variable) and "spike" (release an output) according to conditions on both time and accumulated inputs. Although the proposed models do not manage mimic every kind of neuron behaviors, the authors seemed to find the results encouraging. The modeling using TAs has replaced previous ones, that came mostly from analysis (differential equation models and hybrid models) and statistics (more 'usual' neural networks).

More closely related to learning, some researchers have tried to use formal learning to combine learning, abstraction, test and modeling while keeping an expert user in the center of the acquisition loop. This has been applied to the active learning of large scale software such as commercial games [XSH+05], which the present the interest to have a loosely defined 'enjoyability' goal, that can only be evaluated by experts. This makes the learned model central for visualization purposes and calls for an interactive method instead of a monolithic tool. The authors focus on rule-based learning (trying to learn implications out of a scenario). While the approach and the presence of a working implementation show the wide applicability of the model learning methods, the article unfortunately lacks the space to conduct an interesting technical development.

Model learning is also used in medicine. This is not new, with the important use of Markov's models. In a recent thesis [Sch13] the modeling is pushed further with the addition of an explicit modeling of time. The author use probabilistic real time automata (and subclasses of these) to model the evolution of a disease, taking into account both timing and discrete (the different symptoms) aspects.

1.2 Algorithms

1.2.1 Overview

This part is focused on the algorithms of the aforementioned articles. They are compared based on some qualitative criteria in the table Fig. 2 where A/P/pP stands for active passive or passive from positive examples only learning, and the models abbreviations refer to A for automata, D for deterministic, P for probabilistic, T for timed, RT for real time, L for language, R for regular and MM for Mealy machines with timers.

In the table, note that the active learning algorithms are not incremental as a whole, because they terminate only with a correct model, but the partial hypothesis can be

Algo	On/Off-line	A/P/pP	Incremental	Model	Data
ALERGIA	Off	pP	No	PDA	No
MDI	Off	pP	No	PDA	No
BUTLA	Off	pP	No	PTA	No
OTALA	On	pP	Yes	PRTA	No
k-Tail	Off	pP	No*	DA	No
gk-Tail	Off	pP	No	DA	Yes
Tk-Tail	Off	pP	No*	TA	No
TAUTOKO	Off	P	Yes	DA	Yes*
TREM	Off	pP	No	TL	No
L*	On	A	No	RL	No
[JV17]	On	A	No	MMT	No

Figure 2: Some of the mentioned algorithms and their characteristic

reused in a incremental way. Similarly the method marked with an asterisk could be adapted to be incremental.

In TAUTOKO the data are in fact stored in the set of traces and not taken in the model.

1.2.2 Offline state merging principle

The general principle of the offline passive learning from positive samples algorithms by state merging - ALERGIA MDI BUTLA on the black box side, the variations on k-Tail on the software side- is presented, and the two main operations are briefly discussed.

Computation of the PTA Given a finite set S of (finite) words of an unknown language $\mathcal{L}$, a tree is constructed, generally either as a collection of the traces of S or its prefix tree acceptor (PTA). The prefix tree acceptor is a deterministic tree simply recording all the different executions at the same time. An example is given in Fig. 3. The treatment of the timed component of the behaviours, when it exists, depends on the

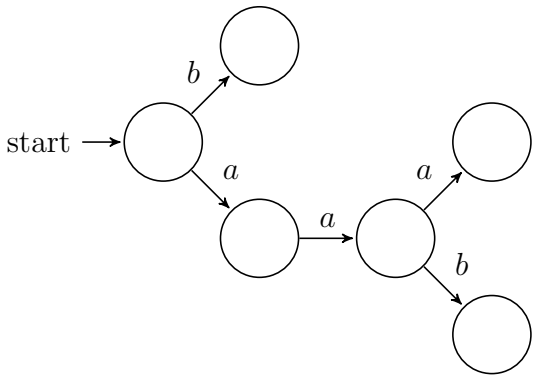

Figure 3: PTA for $S = \{a, b, aaa, aab\}$

article. Some create a branch for every execution [PMM17], while other construct the tree without splitting on temporal behaviour and record the clock values on top of it.

Generalization Once the tree is computed, the model is generalized by merging states that are "close". The way to measure this changes between algorithms. It can be general (KL divergence...) or local (k-future, subtrees...). In the case of timed models, it is also important to split merged states with respect to the timing information. This is generally done according to the differences of observed behaviors for different timing, or to the different modes of the statistical estimation of the timing.

In Fig. 4, the automaton obtained after a 1-future merge of the PTA displayed in Fig. 3.

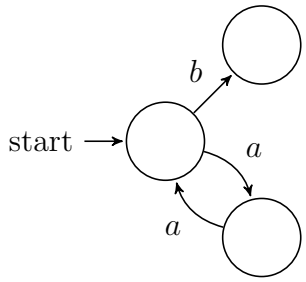

Figure 4: Automaton obtained after a 1-future merge

1.3 Theoretical results

The complexity of he models that can be learned in the most used settings has been intensively studied. Gold [Gol67] showed that primitive recursive languages can be identified in the limit by passive learning (with both positive and negative examples) while only only finite languages can be identified from text (positive examples only) in the limit.

Definition 1.2: Identification in the limit

A language $\mathcal{L}$ is said to be identifiable in the limit by a language learnability model if this models contains an algorithm g that converges in finite time to $\mathcal{L}$ if an allowable training sequence is provided.

The definition of an "allowable" learning sequence vary depending on the precise learning framework.

Gold then investigated the learning from a *finite* set of given data, and found that DFA learning[2] is NP-hard in this framework [Gol78]. Angluin gave a characterization of the learnability from text [Ang80], and investigated some methods to extend the class of learnable languages. This led to the active learning and L* algorithm [Ang87] and to the stochastic learning from text [Ang88].

The PAC framework was first proposed by Valiant in [Val84].

More references can be found in [CT04] for PAC learnability and in [Gri08] for active learning.

[2]and minimal DFA learning

1.4 Models

The theoretical restrictions aforementioned have a direct consequence on the models used by the learning algorithms. I concentrate here on finite state machines, as they occupied the most of the bibliography.

First, a great part of the models are deterministic, those that are not usually have a statistical reason to introduce non-determinism (splitting on two overlapping modes) or relies on the approximations made on the time. Furthermore, a great number of models are probabilistic, as this increases the modeling abilities.

In order to model the time, most model restrict themselves to an unique clock (absolute time or real-time automata), or (with the help of active learning or some insight on the clock structure) extends to event recording. The main exception to this rule could be the Mealy machines with timers, thanks to the equivalence of expressivity between their timed and untimed behaviors.

2 Bibliography

Remarque The citation for Jonsson and Vaandrager 2018 [JV17] is almost empty... this articles does not appear anywhere and have no doi...

References

[Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980. DOI: 10.1016/S0019-9958(80)90285-5. URL: https://doi.org/10.1016/S0019-9958(80)90285-5.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. DOI: 10.1016/0890-5401(87)90052-6. URL: https://doi.org/10.1016/0890-5401(87)90052-6.

[Ang88] Dana Angluin. Identifying languages from stochastic examples. Technical report, 1988.

[BF72] Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, 21(6):592–597, 1972. DOI: 10.1109/TC.1972.5009015. URL: https://doi.org/10.1109/TC.1972.5009015.

[CDD17] Giovanni Ciatto, Elisabetta De Maria, and Cinzia Di Giusto. *Modeling Third Generation Neural Networks as Timed Automata and verifying their behavior through Temporal Logic*. PhD thesis, Université Côte d'Azur, CNRS, I3S, France, 2017.

[CO94] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications*, pages 139–152, 1994. ISBN: 978-3-540-48985-6.

[CT04] Alexander Clark and Franck Thollard. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, 2004. URL: http://www.ai.mit.edu/projects/jmlr/papers/volume5/clark04a/clark04a.pdf.

[CW98] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998. DOI: 10.1145/287000.287001. URL: http://doi.acm.org/10.1145/287000.287001.

[DKM+10] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Sebastian Hack, and Andreas Zeller. Generating test cases for specification mining. In *Proceedings of the Nineteenth International Symposium on Software Testing and Analysis, ISSTA 2010, Trento, Italy, July 12-16, 2010*, pages 85–96, 2010. DOI: 10.1145/1831708.1831719. URL: http://doi.acm.org/10.1145/1831708.1831719.

[DKM+12] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Gordon Fraser, Sebastian Hack, and Andreas Zeller. Automatically generating test cases for specification mining. *IEEE Trans. Software Eng.*, 38(2):243–257, 2012. DOI: 10.1109/TSE.2011.105. URL: https://doi.org/10.1109/TSE.2011.105.

[Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. DOI: 10.1016/S0019-9958(67)91165-5. URL: https://doi.org/10.1016/S0019-9958(67)91165-5.

[Gol78] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978. DOI: 10.1016/S0019-9958(78)90562-4. URL: https://doi.org/10.1016/S0019-9958(78)90562-4.

[Gri08] Olga Grinchtein. *Learning of Timed Systems*. PhD thesis, Uppsala University, Sweden, 2008. URL: http://nbn-resolving.de/urn:nbn:se:uu:diva-8763.

[JV17] Bengt Jonsson and Frits W. Vaandrager. Learning mealy machines with timers. In 2017.

[KV94] Michael J. Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, August 1994. ISBN: 0262111934. URL: http://www.worldcat.org/isbn/0262111934.

[LMP08] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. Automatic generation of software behavioral models. *2008 ACM/IEEE 30th International Conference on Software Engineering*:501–510, 2008.

[Mai14] Alexander Maier. Online passive learning of timed automata for cyber-physical production systems. *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*:60–66, 2014.

[MNE15] Alexander Maier, Oliver Niggemann, and Jens Eickmeyer. On the learning of timing behavior for anomaly detection in cyber-physical production systems. In *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015) co-located with 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (Safeprocess 2015), Paris, France, August 31 - September 3, 2015.* Pages 217–224, 2015. URL: http://ceur-ws.org/Vol-1507/dx15paper28.pdf.

[MNJ+11] Alexander Maier, Oliver Niggemann, Roman Just, Michael Jäger, and Asmir Vodencarevic. Anomaly detection in production plants using timed automata - automated learning of models from observations. In *ICINCO 2011 - Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics, Volume 1, Noordwijkerhout, The Netherlands, 28 - 31 July, 2011*, pages 363–369, 2011.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[NL15] Oliver Niggemann and Volker Lohweg. On the diagnosis of cyber-physical production systems: state-of-the-art and research agenda. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 2015. URL: http://dl.acm.org/citation.cfm?id=2888116.2888294.

[PMM17] Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani. Timed k-tail: automatic inference of timed automata. *CoRR*, abs/1705.08399, 2017. URL: http://arxiv.org/abs/1705.08399.

[RS89] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences (extended abstract), 1989.

[Sch13] Jana Schmidt. *Machine learning of timed automata.* PhD thesis, Technical University Munich, 2013. URL: http://nbn-resolving.de/urn:nbn:de:bvb:91-diss-20131216-1145664-0-4.

[SNF17] Lukas Schmidt, Apurva Narayan, and Sebastian Fischmeister. TREM: a tool for mining timed regular specifications from system traces. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 901–906, 2017. DOI: 10.1109/ASE.2017.8115702. URL: https://doi.org/10.1109/ASE.2017.8115702.

[TDH00] Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using kullback-leibler divergence and minimality. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 975–982, 2000.

[Val84] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. DOI: 10.1145/1968.1972. URL: http://doi.acm.org/10.1145/1968.1972.

[VWW08] Sicco E Verwer, Mathijs M de Weerdt, and Cees Witteveen. Efficiently learning simple timed automata. *Induction of Process Models*:61–68, 2008.

[WLN17] Stefan Windmann, Dorota Lang, and Oliver Niggemann. Learning parallel automata of plcs. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017, Limassol, Cyprus, September 12-15, 2017*, pages 1–7, 2017. DOI: 10.1109/ETFA.2017.8247693. URL: https://doi.org/10.1109/ETFA.2017.8247693.

[WML02] John Whaley, Michael C. Martin, and Monica S. Lam. Automatic extraction of object-oriented component interfaces. In *Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2002, Roma, Italy, July 22-24, 2002*, pages 218–228, 2002. DOI: 10.1145/566172.566212. URL: http://doi.acm.org/10.1145/566172.566212.

[XSH+05] Gang Xiao, Finnegan Southey, Robert C. Holte, and Dana F. Wilkinson. Software testing by active learning for commercial games. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 898–903, 2005. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-142.php.