Self-learning assessment of communication in distributed embedded systems – a feasibility study

Falk Langer, Erik Oswlad

Fraunhofer ESK, Hansastrasse 32, Munich, Germany
{falk.langer, erik.oswald}@esk.fraunhofer.de

Abstract. This paper addresses the problem of evaluating the communication behavior of cyber physical systems. An important problem for the validation of the interaction in the distributed system is missing, wrong or incomplete specification. In this paper, the application of a new approach for assessing the communication behavior based on reference traces is presented and evaluated. The benefit of the approach is that it works automatically, with low additional effort and without using any specification. This paper provides a use case in conjunction with a feasibility study to investigate the applicability of a self-learning anomaly detection methodology. The data of the feasibility study are created by applying the described anomaly detection within a real vehicle network.

Keywords: embedded system validation, testing procedures, network trace analysis, self-learning test methods

1 INTRODUCTION

This paper focuses on test and validation of the communication behavior in cyber physical systems (CPS). On such systems with highly distributed functionality like it can be found in modern car's electronics, the communication behavior is an important aspect on system validation. At field operational test it is important to analyze the network traffic in a fully assembled car. Even if all single electronic control units are tested exhaustively, a significant portion of remaining bugs resulting in errors or malfunction is lately found at real driving or field operational test.

The most important problem of ensuring the correct interaction of functions in CPS at system-level is missing, wrong or incomplete specification (compare [11] and [4]). There are many works of research in progress that tries to improve the process of creating system specification, with the goal of building better test cases for validating the communication on system level. Nevertheless it is still an extensive process to get sufficient test models.

Because network traffic represents the internal behavior of a distributed system, its analysis can help to detect possible bugs earlier and faster. But especially on system level test it is not easy to rate about the correctness of communication at the network. In [10], a new approach for building observer models to evaluate communication behavior automatically, with low additional effort and without using any specification

was introduced. There it was shown that is possible to infer meaningful automata from a network-reference trace. To check the applicability of the proposed approach this paper provides a feasibility study that shows the integration of the methodology proposed in [10] in an existing test scenario and examines the quality of the automata for detecting bugs within the communication behavior.

The paper is structured as follows. In chapter 2 the use case for the proposed methodology and its integration in the test process explained. Chapter 3 provides the previous work and basically describes the new self-learning methodology. In chapter 4 the quality measurements for proofing the feasibility are defined and calculated. The paper closes with chapter 5 conclusion and future work.

2 INTEGRATION TO FIELD TEST

The new approach presented in this paper shall help to find bugs in field operational tests faster. For this reason an important aspect of the proposed solution is the integration in the established testing and validation process for the car's electronic infrastructure. To get an overview about the testing process of this distributed but even closed system in the following the basic testing steps for such a distributed system are described.

As in every software development cycle the first test stage are unit tests and basically the second stage are integration tests. On integration tests, the different applications belonging to an electronic control unit (ECU) are integrated and the basic functionality required from this ECU is tested. The next test level in testing can be characterized as system validation, often it is called system test. This is a test on system level, where the interaction of different ECUs is tested.

Because of the strong interaction of the embedded applications of a car with its environment, field operational tests are commonly finalizes the validation. For software realized functions this kind of test became important at least with the introduction of advanced driver assistant systems which need to be evaluated in real driving tests (compare [5]). Within this background it is a well-established practice that even for the cars electronic infrastructure an endurance test as final acceptance test is executed. This endurance test is performed within real driving field operational tests.

Since the distributed network of ECUs inside the car is a closed system, in most cases it is not possible to control the endurance test on network level. In case of field operational test, mostly normal driving tasks are executed by the test drivers. At this testing level the test driver is only able to detect software bugs that lead to a noticeable malfunction of the car and its components to the driver. Because this is a very limited perspective to the executed software system, in most cases the network traffic from inside the car is recorded at test drives. The recorded network traffic provides information about the internal behavior of the cars electronic infrastructure at test drives. These traces are beside the voting of the test driver, the only source of information for validating the behavior of the cars electronic.

Only when these test drives are executed without a detected malfunction over a dedicated distance of kilometers the electronic system of the tested car passed the final acceptance test. There are still a lot of remaining bugs that are lately found

within these tests with fully assembled car. It did not surprise that there could be easily summate a few million kilometers until all remaining bugs are found, fixed and a test period can be successful completed.

There are two ways to shorten this expensive and time consuming procedure to get a fault free tests drive period within the endurance test. The first one is to reduce the number of remaining bugs that can cause malfunction at test drives. This require better testing methodologies at earlier development phases. Or secondly try to identify possible bugs faster and more efficient within the test drives.

The first one is questionless methodically the clean way. But one of the basic problems in praxis is missing, wrong or incomplete specification of system requirements. This problem becomes commonly relevant with a high number of interacting functions like it can be found highly distributed functionality. To solve this problems there are many research in progress (e.g. [2],[3])

This paper provides a new solution for the second way, the faster identification of potential bugs within test drives. This solution basically tries to identify changes in the system behavior by comparing it with a reference trace. Thereby the probability that this new behavior which is not represented within the reference trace is caused by a bug seems to be high. For this reason a self-learning methodology for an automatic evaluation of the recorded network traffic from field operational tests is presented and evaluated in this paper.

3 PREVIOUS WORK

This chapter provides an overview of the author's previous work that motivates this new approach and provides the foundation and experiments for the proposed self-learning method.

In [9], the idea of extracting dependency models from qualified communication behavior to rate the communication of further test cases, was motivated. With this idea a new methodology for assessing the communication behavior in regions with incomplete or missing specification should be better testable.

Therefore the goal is the construction of a method that allows a qualitative comparison between communication that is presented within a reference trace and newly recorded traces. The essential outcome of the proposed procedure is the awareness, that the newly recorded network trace represents a new system behavior, which was not represented within the reference trace. If such a behavior is recognized, the method outputs a trigger or some equivalent information to the tester. At this point two potential expectations about the tested network behavior can be made: 1) A newly implemented or just jet not observed behavior was found, or 2) A bug in in communication behavior is detected. Just at this point a system expert has to decide if the proposed method detects case 1) or 2). Surely it is not possible to detect bugs, which are already within the reference trace included, but if no other tests detect these bugs and these bugs did not lead to malfunction, it is not sure if it is a bug or just unspecified behavior.

In conjunction to this idea, starting with [8], the learning problem of extracting behavior models from traces was considered. In [8] it was pointed out that basically within a recorded network trace no sequences boundaries are visible. Instead a trace is

one single but very long sequence, which is a challenge for most learning algorithms. With a first simple system hypothesis, where a trace is a stream of events that can be generated from a finite state automaton, the applicability of an artificial neural network was examined within [8].

Because of the unsatisfactory false alarm rate and the high performance utilization of the neural networks, other learning algorithms were looked for. The Angluin L* algorithm which is able to generate finite state automata (compare [1] and [13]) seems to be a good candidate for inferring reliable dependency models. In [7] the adaption of the Angluin L* learning algorithm to learning automata from network streams was shown. The result of the learning process from L* are acceptance automata. It can be shown that these automata describe very accurate the behavior of a given reference sequence of events, without false alarms and with a maximum on inferable dependencies. But at the evaluation with a real car network trace even the L* Algorithm fails to infer reasonable automata. The finding has to be made, that the learning problem is too much complex for the algorithm.

A solution for this problem was provided in [10]. There a methodology is provided that reduces the complexity of the learning task, by separating sub-traces which describe independent execution graphs. With this solution it is possible to infer an acceptance automaton for each sub-trace that describes the behavior of the events in this sub-trace satisfactory.

The identification of independent execution graphs within the network trace was undertaken in [10] with a new clustering approach based on a spectral analysis. It could be shown that there is a high probability that events with similar behavior in time belonging to the same execution graph. In [10] the evaluation of the clustering methodology was done by the application to a real car network trace taken from a controller area network (CAN) that connects the powertrain ECUs. At the result approximately 70% of the behavior of the CAN trace is covered by the inferred acceptance automata. This research results show that it is possible to set up an unsupervised self-learning methodology that infers behavior models from a network trace without the usage of a specification.

If the reference trace represents the normal behavior of a system the inferred automata should accept this normal behavior. If a newly recoded trace holds even a normal behavior of the system, the automata should accept this trace. If this trace holds another behavior it should be rejected by the automata. If the inferred automata do not accept a newly recorded trace, this trace potentially holds a behavior that is out of the norm. This can be called an out of norm (OoN) behavior (comp. [12]) of the trace. If an OoN-behavior is detected the tester will be informed by a OoN-trigger. The above explained methodology for inferring acceptance automata to use them to evaluate newly recorded traces will be called OoN-detection in the following.

4 QUALITY MEASUREMENTS

Even though it is now possible to extract acceptance automata that describe the behavior of the reference trace satisfactory, the usage of these automata for evaluating other traces is still not proofed. For applying this methodology to the introduced use case of finding bugs within network traces from field operational test, it is necessary

to proof the quality of the inferred automata in conjunction to that use case. That means, that a more detailed analysis of the expectable rate of false alarms and the percentage of bugs that are detectable needs to be done.

In the following the basic characteristics for determining the quality and usability of the OoN-detection are explained, defined and estimated. With the usage of real CAN-traces the feasibility of the proposed OoN-detection in an automotive test scenario is examined.

4.1 The coverage criteria

The first criterion for a test mechanism is the test coverage. In the case of the proposed self-learning approach the definition of coverage needs to be adopted to the visible system behavior. The visible system behavior in this case is completely included within the reference trace. The coverage needs to be calculated in relation to the reference trace. A trace basically consists of a set of events $e_i \in \mathcal{E}$. A sequence of events is defined by $s(e \in \mathcal{E}) = e_1, e_2 \dots e_n \mid e \in \mathcal{E}$. In case of a network trace a sequence defines the complete recorded amount of events in the given period, which can be easily more than 10^6 events.

In [10] it was shown that is not possible to infer for all events $\mathcal{E}$ acceptance automata. That leads to the effect that only a sub set of the elements of $\mathcal{E}$ are mentioned by the OoN-detection. That subset of $\mathcal{E}$ leads also to a subset of the trace and of sequence s, because some single events of s are missing $\mathcal{E}^* \subseteq \mathcal{E} \rightarrow s(e \in \mathcal{E}^*) \subseteq s(e \in \mathcal{E})$. This results in two different coverage measurements. These are the relation the event coverage C_e and the trace coverage C_t with:

$$\text{event coverage} \qquad C_e = \mathcal{E}^*/\mathcal{E} \tag{1}$$

$$\text{trace coverage} \qquad C_t = |s(e \in \mathcal{E}^*)|/|s(e \in \mathcal{E})| \tag{2}$$

4.2 The classification criteria

The essential criterion for classification is the rating if the classified belongs to a specific class of object or not. In case of the OoN-detection the trace needs to be classified in the two decision classes: (1) the trace has the same behavior as the reference trace (2) the trace has a different behavior as the reference trace. To rate if the decision of the OoN-detector is correct there are for different results possible:

1. True positive (tp) :The trace was classified to has the same behavior as the reference trace and this was correct
2. False positive (fp):The trace was classified to has the same behavior as the reference trace, but this was not correct
3. True negative (tn):The trace was classified to has a different behavior as the reference trace and this was correct
4. False negative (fn):The trace was classified to has a different behavior as the reference trace, but this was not correct

With this attributes the two important relational criteria, the false alarm rate $P(fn|tp)$ and the rate of detectable anomalies $P(tn|fp)$, can be calculated.

false alarm rate	$P(fn\|tp) = \frac{fn}{tp+fn}$	(3)
rate of detectable anomalies	$P(tn\|fp) = \frac{tn}{tn+fp}$	(4)

Estimating the rate of detectable anomalies.

The rate of detectable anomalies determines the percentage the method find real OoN deviations in a trace. The optimal way to calculate this rate would be to have traces that holds known bugs, that can be presented to the OoN-detection. But for this kind of bug detection no public data sets are available.

A practicable method for estimating the rate of detectable anomalies is the instrumentation of a trace which has no bugs inside and is completely accepted by the inferred automata. In the proposed use case this needs to be the reference trace that was used to learn the acceptance automata. For estimating the rate of detectable anomalies the reference trace was instrumented in three different ways: (1) a randomly selected event is deleted from the trace (2) a randomly selected event is duplicated and (3) a randomly selected event is replaced by another randomly selected event.

Estimating the false alarm rate.

The estimation of the false alarm rate turns out to be more complicated. Since the reference trace describes not all possible behavior, the OoN-detection will find necessarily new behavior within the test traces. But without a deeper knowledge of the system, it is not possible to decide if the OoN-detection did in fact identify new behavior or not.

Because a false alarm is not directly identifiable it would be helpful to have a look at the causes of false alarms. If it is clear what circumstances can cause a false alarm, it should be possible to estimate the count of false alarm otherwise.

The inferred acceptance automata of the OoN-detection mechanism are not based on probabilistic decisions like neural networks or Markov chains. Therefore it can be shown that the inferred acceptance automata do always evaluate the reference trace correct and in the same way. But these acceptance automata can be overfitted that they only accept the behavior of the reference trace. An overfitting in that case means that an acceptance automaton evaluates events that are not correlated to each other. Only if the events are correlated, they can have a dedicated behavior that is reproducible and can therefore declared as normal. If an inferred acceptance automaton is overfitted it accepts only the reference trace but no other traces. It can be pointed out that the most significant part of false alarms will be caused by overfitted automata. Therefore the ascertainment of the number of overfitted automata will give a first impression of the false alarm rate.

It is a strong indication of overfitting, if an inferred acceptance automaton rejects all the tested traces except the reference trace. To estimate the false alarm rate, one test trace is evaluated by the inferred acceptance automata. All automata that reject

this trace are taken in the next step to evaluate the other test traces. If an automata reject all test traces it is very likely overfitted.

5 Results of feasibility study

The best way to examine the feasibly of the described OoN-detection mechanism would be the evaluation with real network data that have known anomalies. But for the proposed use case, for the evaluation of network data from a cars network, there are no such data sets available.

The evaluation results presented in this chapter are generated by using real network data that are taken from a powertrain CAN of a car in series production. Because of the usage of a proven in use product, it is expected that these network data do not have any faults or anomalies.

The evaluation data set.

The starting point for the evaluation is the reference trace that was used in [10] to infer acceptance automata. This reference trace has a length of 12.80 minutes with a sequence length of $2{,}5 * 10^6$ events that is described by a set of 7172 different events. Since this reference trace is recorded at a car in series production it is estimated that it has no bugs inside. From this reference trace 2,848 different acceptance automata was inferred with the methodology explained in [10].

For the quality measurement four test traces on the same car at different driving scenarios are recorded. The recording time of these traces is in the range of 14.60 min to 2.85 min and has in total approximately 26 min (see Table 1).

Table 1. Length of reference and test traces

Reference	Trace 1	Trace 2	Trace 3	Trace 4
12.80 min	14.67 min	6.38 min	2.85 min	3.07 min

False alarm rate.

For estimating the false alarm rate Trace 1 is evaluated by the 2,848 acceptance automata. This trace is rejected by 1,943 automata as shown in the first column of Fig. 1. When these automata are checked for overfitting with trace 2 a number approximately $P(\mathrm{fn}|\mathrm{tp}) = 90\ \%$ seems to be overfitted because they reject trace 1 and 2. If these overfitted automata are excluded 181 remaining automata generate an OoN-trigger. If these remaining automata are additionally checked with trace 3 and 4 a number of 114 automata are left that seems to be not overfitted. That means with the additionally plausibility check with only 2 trace a remaining false alarm rate of $P(\mathrm{fn}|\mathrm{tp}) = 40\ \%$ can be achieved (compare Fig. 1).

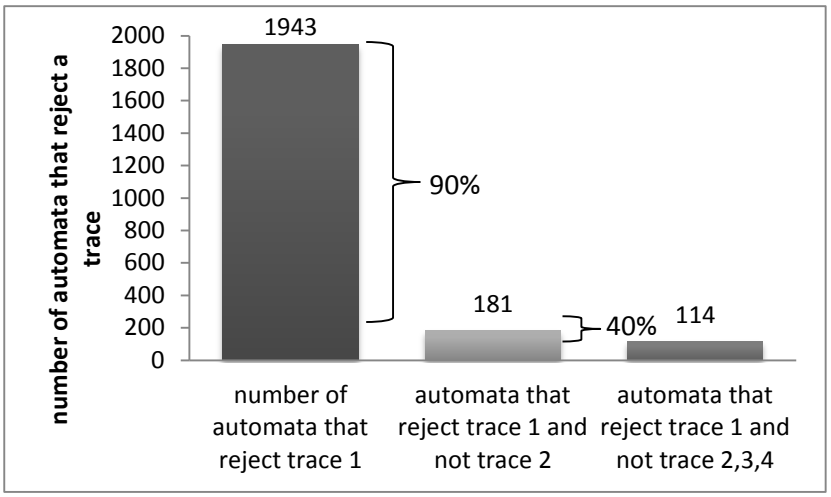

Fig. 1. Number of automata that reject the test traces

Rate of detectable anomalies.

The experiments for estimating the rate of detectable anomalies are executed by instrumenting the reference trace. The results are shown in Table 2. In the experiment approximately 0.1 % of the trace was modified, that leads to about 2.500 injected anomalies.

Table 2. Experimental results for rate of detectable anomalies

Instrumentation	(1) deleted	(2) duplicated	(3) replaced
Rate of detectable anomalies $\boldsymbol{P(tn\|fp)}$	94 %	95 %	91 %

Coverage.

The 2.848 initial inferred automata cover approximately 80 % of the trace (see Fig. 2 with $C_e = 88\,\%, C_t = 77\,\%$). When the overfitted automata are excluded a number of 770 acceptance automata are usable for OoN-detection. These do have still a coverage of approximately 45 % (compare Fig. 2 right column).

Coverage OoN-detection – normal and not normal behavior.

After the overfitting analysis a number of 770 automata are usable for OoN-detection. If the OoN-detection mechanism is applied to these automata for the given reference traces 621 of these automata do accept all four test traces. A number of 149 automata rejects at least one test traces. In Fig. 3 the resulting coverage of the detected normal behavior and the detected *not* normal behavior is shown.

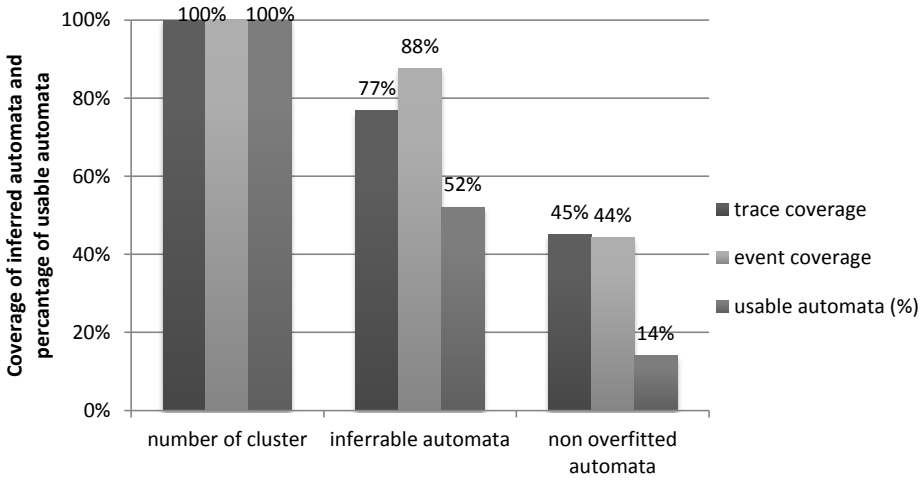

Fig. 2. Coverage criteria calculated for the reference trace

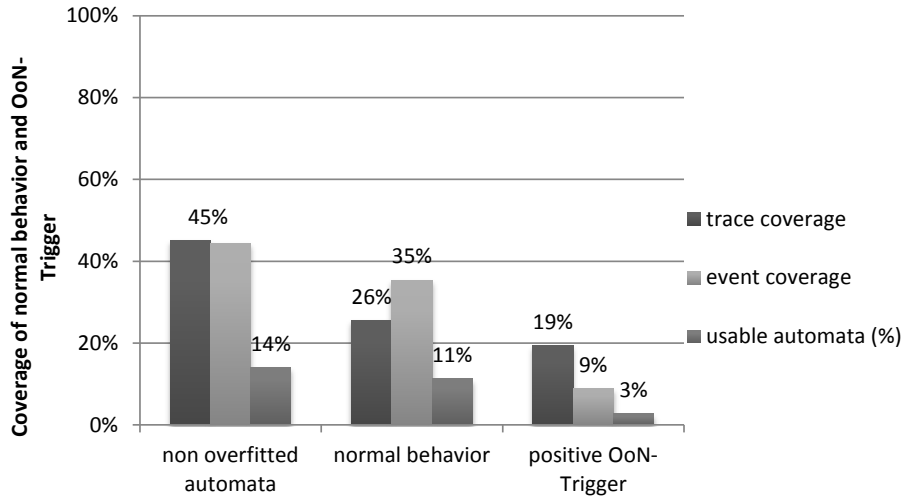

Fig. 3. Coverage criteria calculated for the reference trace

5.1 Discussion of results

The results provide a first impression of the applicability of the proposed self-learning OoN-detection. The evaluation based on real CAN-communication is a first step for approving the proposed OoN-detection. The most critical point is of course the false alarm rate. Although the estimation of this rate tends to be difficult task, the results provide an evidence that the false alarm rate tends to be in the range of 40 %. In comparison to automatic test methods like code checker this seems to be acceptable

(comp. [6]). A very good result was reached by the detection of anomalies with a range of about 90 %. A comparison of the coverage is difficult, but for an unsupervised self-learning approach a system coverage of about 45 % seems to be an excellent result.

6 CONCLUSION AND FUTURE WORK

This paper explains and evaluates an application use case for an self-learning OoN-detection that was established in prior work. Since the proposed OoN-detection is a new approach in the area of system validation, this paper provides a basic quality measurement for this automatic self-learning approach.

It could be shown that the proposed self-learning OoN-detection is potentially usable to detect anomalies in the communication behavior in comparison to a reference trace. Essentially for the critical false alarm rate the usage in a testing environment with approximately 40 % seems to be acceptable.

The provided evaluation results are recorded from a car in series production that has potentially no bugs inside. For this reason a further investigation within a real testing environment needs to done to learn more about the expected outcome and usability of the proposed self-learning OoN-detection for CPS.

7 REFERENCES

[1] Angluin, D. 1987. Learning regular sets from queries and counterexamples. Information and computation 75, 87–106.

[2] Bollig, B., Katoen, J.-P., Kern, C., and Leucker, M. 2007. Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning. In Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems. Lecture Notes in Computer Science. Springer, Braga, Portugal, 435–450.

[3] Drabek, C., Pramsohler, T., Zeller, M., and Weiss, G. 2013. Interface Verification Using Executable Reference Models: An Application in the Automotive Infotainment. In Proceedings of the 6th International Workshop on Model Based Architecting and Construction of Embedded Systems, Miami, Florida, USA, 7:1–10.

[4] Ebert, C. and Jones, C. 2009. Embedded Software: Facts, Figures, and Future. IEEE Computer 42, 4, 42–52.

[5] Goralczyk, M., Schaeufele, B., and Radusch, I. 2011. Logging Design for Vehicle Communication Field Operational Tests. In FAST-Zero'11 Proceedings, 1–6.

[6] Kremenek, T., Ashcraft, K., Yang, J., and Engler, D. 2004. Correlation exploitation in error ranking. SIGSOFT Softw. Eng. Notes 29, 6, 83–93.

[7] Langer, F., Bertulies, K., and Hoffmann, F. 2011. Self Learning Anomaly Detection for Embedded Safety Critical Systems. In Schriftenreihe des Instituts für Angewandte Informatik, Automatisierungstechnik am Karlsruher Institut für Technologie. KIT Scientific Publishing, 31–45.

[8] Langer, F., Eilers, D., and Knorr, R. 2009. Fault detection in discrete event based distributed systems by forecasting message sequences with neural networks. In KI 2009: Advances in Artificial Intelligence. Springer, 411-418.

[9] Langer, F. and Prehofer, C. 2011. Anomaly detection in embedded safety critical software. In International Workshop on Principles of Diagnosis (DX) , 163–166.

[10] Langer, F. and Oswald, E. 2014. Using Reference Traces for Validation of Communication in Embedded Systems. In ICONS 2014, The Ninth International Conference on Systems, 203–208.

[11] Lutz, R. R. 1993. Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. In Proceedings of the IEEE International Symposium on Requirements Engineering, 126-133.

[12] Peti, P., Obermaisser, R., and Kopetz, H. 2005. Out-of-norm assertions [diagnostic mechanism]. In Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE. IEEE, 280–291.

[13] Berg, T., Jonsson, B., Leucker, M., and Saksena, M. 2005. Insights to Angluin s Learning. Electronic Notes in Theoretical Computer Science 118, 3–18.