

IT Licentiate theses  
2006-003

# Regular Inference for Reactive Systems

THERESE BERG

UPPSALA UNIVERSITY  
Department of Information Technology







UPPSALA  
UNIVERSITET

**Regular Inference for  
Reactive Systems**

BY  
THERESE BERG

April 2006

DIVISION OF COMPUTER SYSTEMS  
DEPARTMENT OF INFORMATION TECHNOLOGY  
UPPSALA UNIVERSITY  
UPPSALA  
SWEDEN

Dissertation for the degree of Licentiate of Technology in Computer Science  
at Uppsala University 2006

# Regular Inference for Reactive Systems

*Therese Berg*

Therese.Berg@it.uu.se

*Division of Computer Systems  
Department of Information Technology  
Uppsala University  
Box 337  
SE-751 05 Uppsala  
Sweden*

<http://www.it.uu.se/>

© Therese Berg 2006

ISSN 1404-5117

Printed by the Department of Information Technology, Uppsala University, Sweden

# Abstract

Models of reactive systems play a central role in many techniques for verification and analysis of reactive systems. Both a specification of the system and the abstract behavior of the system can be expressed in a formal model. Compliance with the functional parts in the specification can be controlled in different ways. Model checking techniques can be applied to a model of the system or directly to source code. In testing, model-based techniques can generate test suites from specification. A bottleneck in model-based techniques is however to construct a model of the system. This work concerns a technique that automatically constructs a model of a system without access to specification, code or internal structure. We assume that responses of the system to sequences of input can be observed. In this setting, so called *regular inference* techniques build a model of the system based on system responses to selected input sequences.

There are three main contributions in this thesis. The first is a survey on the most well-known techniques for regular inference. The second is an analysis of Angluin's algorithm for regular inference on synthesized examples. On a particular type of examples, with prefix-closed languages, typically used to model reactive systems, the required number of input sequences grow approximately quadratically in the number of transitions of the system. However, using an optimization for systems with prefix-closed languages we were able to reduce the number of required input sequences with about 20%. The third contribution is a developed regular inference technique for systems with parameters. This technique aims to better handle entities of communications protocols where messages normally have many parameters of which few determine the subsequent behavior of the system. Experiments with our implementation of the technique confirm a reduction of the required number of input sequences, in comparison with Angluin's algorithm.



# Acknowledgments

I would like to acknowledge the assistance from a few people. I would first like to express my appreciation of my supervisor Bengt Jonsson. I am grateful for the guidance and encourage he has given me. I also would like to acknowledge my second supervisor Joachim Parrow, for his engagement in my work. I am grateful for the collaboration with Mayank Saksena on the very first project I did, and for the rewarding collaboration with Harald Raffelt. In addition to a pleasant collaboration with Harald, he and his wife Verena showed me great hospitality during my visit at the University of Dortmund. I would like to thank Johan Blom for letting us use a piece of his work for an example in a paper.

I am grateful for the friendship and support from my colleagues Ahmed Rezine, Frederic Haziza, Johann Deneux, Noomene Ben-Henda, and Lisa Kaati at work and off work. I thank Anders Hessel, Olga Grinchstein, and Paul Pettersson for creating an open research environment in the testing group.

At last I would like to thank my family Anna-Karin, Maj-Britt, and Bengt-Olof Berg for their support.



# List of Included Papers

## Paper A

M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

## Paper B

Therese Berg, Bengt Jonsson, Martin Leucker, and Mayank Saksena. Insights to Angluin’s learning. *Electronic Notes in Theoretical Computer Science*, 118:3–18, 2005.

## Paper C

Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines with parameters. In *FASE*, volume 3922 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2006.

## Comments on my Participation

### Paper A

I am the sole author of the second half of Chapter 19, Section 19.4-19.5, and the co-author of the summary, Section 19.6.

### Paper B

Regarding the work in Paper B, I have together with a co-author implemented Angluin’s algorithm and its domain-specific optimization. We have together conducted all experiments and analyzed the result. I am a co-author to Section 4 and 5 in the paper.

### Paper C

In the work that Paper C is based, I have participated in the discussion which lead to the algorithm that is presented in Paper C. I am the principal

author of Section 5, and co-author of Section 6. I have participated in all work concerning the implementation, construction, conducting experiments, and analysis.

## Other Work

- Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In Maura Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2005.
- H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 62–71, New York, NY, USA, 2005. ACM Press.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Research Problems Addressed in This Thesis . . . . .	2
1.3	Contributions . . . . .	2
1.4	Thesis Organization . . . . .	3
<b>2</b>	<b>Regular Inference</b>	<b>5</b>
2.1	Algorithms . . . . .	6
2.2	Complexity . . . . .	7
2.3	Optimizations . . . . .	7
2.4	Equivalence Oracle . . . . .	8
2.5	Regular Inference of Parameterized Systems . . . . .	9
2.6	Regular Inference together with Model Checking . . . . .	9
2.7	Regular Inference in Relation to Conformance Testing . . . . .	10
<b>3</b>	<b>Summary of Papers</b>	<b>13</b>
3.1	Paper A: Learning Finite State Machines . . . . .	13
3.2	Paper B: Insight's to Angluin's Learning . . . . .	14
3.3	Paper C: Regular Inference for State Machines with Parameters . . . . .	15
<b>4</b>	<b>Conclusions and Future Work</b>	<b>17</b>



# Chapter 1

## Introduction

### 1.1 Background

The type of systems that are considered in this thesis are so called *reactive systems*, systems that are usually intended to continuously receive input. Examples of such systems are web servers, communication protocols, operating systems, and controllers in embedded systems. In an introductory phase in the development of such a system the stakeholders express their requirements on the system to the developers. The requirements together with design choices made by the developers form a specification, with which the finished system must comply. For reactive systems a specification can be expressed formally, for instance, in the form of a state machine, message sequence charts (MSC) [16], expressed in the unified modeling language (UML) [23], in the specification and description language (SDL) [17], or in temporal logic [25]. Compliance with the functional parts of the specification can be checked in different ways, of which some assume a formal specification. Model-based techniques can generate test suites from a formal specification [9, 7], and model checking techniques can be applied to a model of the system [7, 14] or directly to source code [29, 13].

A bottleneck in the above mentioned model-based techniques is to construct and maintain an up to date model of the system. System updates or modifications lead to inconsistencies between model and system. An approach to address this problem is to construct models by hand from inspection of code or specification. Models can also be automatically generated by applying techniques that analyze source code [12, 30]. They can also be generated from samples of input/output (I/O) sequences of the system using techniques in *machine learning*.

We have focused our work on a machine learning technique to infer state machines called *regular inference*. We assume that we know what individual inputs can be applied to the system, and that the system we want to model has a deterministic functionality. In order for the regular

inference algorithms to infer a state machine they observe the responses of the system to selected sequences of input. When enough sequences have been collected, the algorithm is able to generate a model. For simplicity, the model only accepts or rejects input sequences. The set of input sequences accepted by the model forms a so called regular language (see Chapter 2 for more details on regular languages). Our goal is to investigate to what extent existing algorithms for regular inference are applicable to construct models of reactive systems and what can be done to improve them.

## 1.2 Research Problems Addressed in This Thesis

We will focus on algorithms for regular inference and specifically three of them. Our goal is to investigate these three algorithms, to describe their differences and resemblances. We also want to present adaptations of regular inference to model checking [24, 11], and to domain-specific applications [15].

The number of I/O sequences required to generate a model can be prohibitive if the time needed to execute each input sequence in the system is long, or the system is large. In such cases the model construction takes too much time to be practical. Hence we are interested in how a common regular inference algorithm, the  $L^*$  algorithm by Angluin [1], performs on large examples and if it performs differently for different types of examples.

When we build a model of an entity of a communication protocol it demands a great effort. The number of input symbols is exponential in the number of parameters in a protocol data unit (PDU) package. As a consequence, the required number of I/O sequences by Angluin's  $L^*$  [1] is exponential in the number of parameters. Hence we aim to construct an algorithm which only requires that input symbols with parameter settings that determine how the system reacts in a state are input to the system. With that all possible parameter setting of a communication message need not be executed.

## 1.3 Contributions

The main contributions of this thesis are the following.

- A survey on the most well-known techniques for regular inference.
- We have implemented the  $L^*$  [1] and showed by experiments with synthesized examples an approximately linear growth, in terms of transitions in the system, of the required number of membership queries (see Chapter 2 for details on membership queries). This clearly indicates problems in the algorithm's practical applicability. On a particular type of examples, with prefix-closed languages (see Chapter 2 for details on this language property), the required number of membership

queries grew approximately quadratically in the number of transitions of the system. However, in an optimization for systems with prefix-closed languages we were able to reduce the number of membership queries with about 20%.

- We have developed a regular inference technique for systems with parameters. Our aim is to better handle entities of communication protocols where messages normally have many parameters of which few determine how the system reacts. The experiments we have performed on our implementation confirm a reduction of the number of membership queries (see Chapter 2 for details on membership queries) with our technique.

## 1.4 Thesis Organization

The thesis is organized as follows. Chapter 2 gives a presentation of regular inference and related work. The following chapter, Chapter 3, gives a summary of each paper included in this thesis together with a small discussion. The last chapter, Chapter 4, presents the conclusions made in this thesis and points out interesting topics for future work. Thereafter follow reprints of the three published research papers.



## Chapter 2

# Regular Inference

In regular inference, systems are viewed as black boxes, i.e., there is no access to specification, code, or internal structure. In order to investigate the functionality of a system we observe the responses of the system to selected sequences of inputs. The response of the system is either to execute on the input or reject it.<sup>1</sup> We assume the system to have a *reset*, which puts the system into its initial state.<sup>2</sup>

In the following consider a finite *alphabet*  $\Sigma$  of *symbols*. A *language* is a subset of  $\Sigma^*$ , the set of finite sequences of symbols, also called *strings*. A *deterministic finite automaton (DFA)* over  $\Sigma$  is a structure  $\mathcal{M} = (Q, \delta, q_0, F)$ , where  $Q$  is a non-empty finite set of *states*,  $\delta : Q \times \Sigma \rightarrow Q$  is the *transition function*,  $q_0 \in Q$  is the *initial state*, and  $F \subseteq Q$  is the set of *accepting states*. The transition function is extended from input symbols to strings of input symbols in the standard way, by defining  $\delta(q, \varepsilon) = q$ , and  $\delta(q, ua) = \delta(\delta(q, u), a)$ .

An input string  $u$  is *accepted* iff  $\delta(q_0, u) \in F$ . The *language* accepted by  $\mathcal{M}$ , denoted by  $\mathcal{L}(\mathcal{M})$ , is the set of accepted input strings. A subset  $\mathcal{R} \subseteq \Sigma^*$  is said to be *regular* if  $\mathcal{R}$  is accepted by some DFA. A language  $\mathcal{R}$  is *prefix-closed* if for every  $w$  in  $\mathcal{R}$ , all prefixes of  $w$  are in  $\mathcal{R}$ . We say that a DFA is prefix-closed if its language is prefix-closed. A minimal prefix-closed DFA has exactly one non-accepting state.

We here try to give a succinct description of the main ideas behind regular inference. We assume that a system in which we are interested can be modeled by a DFA  $\mathcal{M}$ . The problem can now be looked upon as identifying the regular language which is accepted by  $\mathcal{M}$ , denoted by  $\mathcal{L}(\mathcal{M})$ .

In a learning algorithm a so called *Learner*, who initially knows nothing about  $\mathcal{M}$ , is trying to learn  $\mathcal{L}(\mathcal{M})$  by asking queries to a *Teacher* and an

---

<sup>1</sup>One approach to handle systems that produces sequences of output is described by Oliver Niese [22].

<sup>2</sup>Assuming that the system is strongly connected, that is, there is a directed path between every pair of states in the system, a model can be generated without the need of reset [18].

*Oracle.* There are two kinds of queries.

- A *membership query* consists in asking the *Teacher* whether a string  $w \in \Sigma^*$  is in  $\mathcal{L}(\mathcal{M})$ .
- An *equivalence query* consists in asking the *Oracle* whether a hypothesized DFA  $\mathcal{A}$  is correct, i.e., whether  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{M})$ . The *Oracle* will answer *yes* if  $\mathcal{A}$  is correct, or else supply a counterexample  $u$ , either in  $\mathcal{L}(\mathcal{M}) \setminus \mathcal{L}(\mathcal{A})$  or in  $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{M})$ .

The typical behavior of a *Learner* is to start by asking a sequence of membership queries, and gradually build a hypothesized DFA  $\mathcal{A}$  using the obtained answers. When the *Learner* feels that she has built a “stable” hypothesis  $\mathcal{A}$ , she makes an equivalence query to find out whether  $\mathcal{A}$  is correct. If the result is successful, the *Learner* has succeeded, otherwise she uses the returned counterexample to revise  $\mathcal{A}$  and perform subsequent membership queries until arriving at a new hypothesized DFA, etc.

## 2.1 Algorithms

There exist a handful of regular inference algorithms. One of the most widely recognized is  $L^*$  created by Angluin [1]. This algorithm organizes the accumulated information obtained from queries and answers in a so called *observation table*. The observation table regards each string as consisting of a prefix and a suffix. The prefixes are indices of rows and the suffixes indices of columns in the table. The set of prefixes  $U$ , is a prefix-closed set of strings, and the set of suffixes,  $V$ , is a suffix-closed set of strings. The table entry for row index  $u \in U$  and column index  $v \in V$  is the answer to a membership query on  $uv$ . A prefix can be thought of as a string which leads to a state in the system, and a suffix as distinguishing prefixes that lead to different states from another.

There are other algorithms that are rather similar to  $L^*$ . One is an algorithm by Rivest and Schapire [27] that uses a *reduced observation table*. Compared to the observation table, this reduced version stores a smaller portion of queries and answers. The requirement on the row indices is relaxed so that the set is not required to be prefix-closed. A third alternative by Kearns and Vazirani [18], uses a completely different data structure to store information, a binary *discrimination tree*. The nodes of the tree contains suffixes which are used as before to distinguishing prefixes that lead to different states from another. Balcázar et al. has presented a unifying concept from which these algorithms, including  $L^*$ , can be viewed [2]. In essence, the algorithms differ in how many membership queries are required before a model is constructed. Among these three, the  $L^*$  algorithm generally performs the largest number of membership queries before a model is created. But because  $L^*$  collects more information before generating a

model, it is also more likely to produce fewer false hypotheses and thus fewer equivalence queries. The upper bound on the number of equivalence queries is however the same for all three algorithms.

## 2.2 Complexity

The complexity of the algorithms is most often measured in the number of required membership and equivalence queries. The reason for this is that executing a membership query involves interaction with the system, and this is likely to require some time. The observation table, or other data-structure for queries and answers, also needs to be stored; for that we need to allocate memory resources. In the following, let  $n$  be the number of states in of the minimal DFA  $\mathcal{M}$ , let  $m$  be the length of the longest counterexample returned in an equivalence query, and let  $|\Sigma|$  be the size of  $\Sigma$ .

Let us first start with the number of equivalence queries. In  $L^*$ , and the algorithms using a reduced observation table or discrimination tree, the upper bound on the number of equivalence queries is  $n$ .

The upper bounds on the number of membership queries are more diverse for these algorithms. Angluin's  $L^*$  requires  $O(|\Sigma|n^2m)$ , the algorithm that uses the reduced observation table  $O(|\Sigma|n^2 + nm)$ , and the algorithm that uses the discrimination tree  $O(|\Sigma|n^3 + nm)$ . The bounds for membership queries depend on whether answers to queries are stored, (they are not in the discrimination tree case but in the other two they are) and how many membership queries are asked before creating a conjecture.

In practice the three inference algorithms would all perform poorly when applied to large systems according to upper bounds. We illustrate this with an example. In our earlier work [4] the model of an ATM protocol, shipped with the Edinburgh Concurrency Workbench [21], used for experiments has 1715 states and 27 symbols. Assuming that the longest counter-example is 1715 symbols long, the effort of applying Angluin's algorithm to this particular example is asking  $27 \times 1715^2 \times 1715 = 1.4 \times 10^{11}$  membership queries in the worst case. Thus, optimizations are necessary for the regular inference to work well in practice.

## 2.3 Optimizations

A suggestion for optimizations by Hungar et al. [15] is based on the idea that knowledge about the domain of the system can be used to reduce the number of membership queries required by a regular inference algorithm. One optimization exploits that instances of communicating processes may behave in the same way and therefore can be interchanged. A second utilizes the deterministic behavior of a system; I/O systems will always give the same output on a specific sequence of inputs. A third employs knowledge

about reactive systems, modeled as finite state machines with prefix-closed languages. They have evaluated their suggestions for optimizations on 7 examples. With these examples they have accomplished a total reduction in membership queries varying between 87% to 99.8% using all three optimizations.

In paper B [4], we have performed an investigation of the performance of  $L^*$ , based on a selection of small protocols and a large set of randomly generated DFAs. The result from applying  $L^*$  to the examples shows that it is generally harder to infer randomly generated prefix-closed DFAs compared to randomly generated DFAs. We applied the optimization specific to prefix-closed DFA. This led to a 20% reduction of membership queries. The reason for this comparatively smaller reduction is that we used randomly prefix-closed DFA and therefore do not have the first and second system properties described in the previous paragraph.

## 2.4 Equivalence Oracle

In the regular inference setting we require an equivalence oracle. The oracle's job is to either confirm that the suggested conjecture is correct or provide a counter-example. There is however no "magical" oracle that will provide this information for free. The oracle is a theoretical construction to make an idealization of a potentially hard problem, in order to provide a clean setup in regular inference. In practice, ways to provide counter-examples are for instance by monitoring the system and collecting a counter-example whenever the model and system disagrees, by letting a "system expert" evaluate the model, or by testing the system with tests from a so called conformance test suite (see Section 2.7 for details regarding conformance testing). But all of these mimicked oracles have their disadvantages; monitoring may produce a very long counter-example which affects the complexity of the regular inference algorithm negatively, involving a "system expert" makes the regular inference technique only semi-automated and therefore less attractive, and finding a counter-example by executing tests from a conformance test suite is like executing membership queries. The advantage of conformance testing is that it provides a systematic way of achieving an answer to an equivalence query. Let  $k$  be the number of states in the hypothesis of the system, and assume that we have an upper bound,  $l$ , on the number of states of a minimal DFA that models the system. Then if  $l > k$ , by applying the tests in a conformance test suite by Vasilevski and Chow [8, 28] to the system we will find at least one test that the system does not pass. This test constitutes a counter-example as answer of an equivalence query. According to Vasilevski [28], an upper bound for the total length of such a test sequences suite is  $O(k^2 l |\Sigma|^{l-k+1})$ , i.e., it is exponential in the difference between the number of states of the system and the hypothesis.

## 2.5 Regular Inference of Parameterized Systems

The technique of regular inference can be used to construct models of entities of communication protocols. Typically in such protocols symbols consists of a message type together with a number of parameters, each of which can assume several values. Hence, the alphabet of such an entity is exponential in the number of parameters. The large alphabet would make the regular inference technique unattractive since it would negatively affect the required amount of membership queries. If only a few of the parameters determine the behavior of the system, it is desirable to avoid asking membership queries for the parameter settings that have no effect. Therefore a regular inference algorithm that specializes in handling these cases has been developed [5]. This algorithm is implemented in the framework of LearnLib [26], a library for automata and regular inference.

The idea behind our specialized algorithm is to infer, for each state and each message type, a partitioning of input symbols into equivalence classes, under the hypothesis that all input symbols in an equivalence class have the same effect on the state machine. Initially all input symbols of a message type are put in one equivalence class. Whenever such a hypothesis is disproved, equivalence classes are refined. In the extended DFA that is inferred by our algorithm, the transitions are labelled with such equivalence classes. The number of membership queries the algorithm require is bounded by  $O(nkm)$ , where  $k$  is the number of transitions of our extended DFA. The result of applying the algorithm to a large set of randomly generated examples shows that the algorithm requires less membership queries compared to  $L^*$  when the number of transitions in the extended DFA is relatively few.

## 2.6 Regular Inference together with Model Checking

*Model checking* [7, 14] is a technique to automatically check that models of systems have a given property. A technique to model check a black box is presented by Peled et al. [24] by combining regular inference and model checking. The idea of combining the two techniques is further elaborated to a method called *adaptive model checking* [11].

In the setting for adaptive model checking we have access to the system, but seen as a black box, and a property we want to check on the system. The basic idea in adaptive model checking is to apply a regular inference algorithm to create a model of the system and then apply a model checking algorithm to check whether the model satisfies the property. If the property holds, the regular inference algorithm makes an equivalence query with the conjecture. If the oracle replies with a counter-example, it is inserted into the regular inference algorithm and the model is revised.

In the case the property does not hold for the model, the model checker provides an error-trace, a string that when the model traverses it, the model violates the property. The adaptive model checking algorithm inputs the error-trace in the system and observes whether it can be executed by the system or not. If it can be, the algorithm concludes that the system does not satisfy the property, otherwise the error-trace is feed back to the regular inference algorithm as a counter-example which uses it to refine the model.

When the property holds and the oracle confirms that the model is correct, the adaptive model checking algorithm terminates and concludes that the system satisfies the property.

## 2.7 Regular Inference in Relation to Conformance Testing

In all model-based techniques, there is the problem of how to check that the model is an accurate description of the system. In the black box setting, a way to check that the model is equivalent to the system is by a technique called *conformance testing*. A conformance testing technique generates a set of tests,  $T$ , a so called *test suite*. A test,  $t \in T$ , consists of a string and the expected output of the system in response to the string. The system is said to *pass* a test if the actual output is the same as the expected output. We can confirm that a model is correct with respect to the system if the system passes all tests in a conformance test suite, and the system and model satisfies required hypotheses. Examples of required hypotheses are that the model is minimized, the system does not change during testing, the transition function for the model is total, and there is a known upper bound on the number of states in the system. If the system fails to pass a test  $t$ , then  $t$  is a counter-example to the conjecture that the model and system are equivalent.

A usual hypothesis is that the number of states of the system is exactly as many as the number of states of the model. Conformance test techniques that can be used in this setting are the W [8, 28], Wp [10], and Z [19] techniques. The main difference between them is that the Wp and Z techniques generate a smaller test suite compared to the W technique. Test suites generated by these techniques are similar to the set of strings asked for in membership queries by regular inference algorithms. In our earlier work [3] we have performed a closer investigation of the relationship between conformance testing and regular inference. Assume that the model  $\mathcal{A}$  has  $k$  states, and the system  $\mathcal{M}$  has  $l$  states. We show in [3] that the set of strings in an observation table,  $UV$ , by which the  $L^*$  algorithm generates the DFA  $\mathcal{A}$  is also a conformance test suite for  $\mathcal{A}$ , generated by the W technique. Consequently, under the assumption that  $k = l$ , there is actually no need for an equivalence test since we then know that we have a correct

model; this also follows from [1]. Assume that  $\mathcal{M}$  has more states than  $\mathcal{A}$ , i.e.  $l > k$ , then using a conformance test suite as an equivalence oracle in a regular inference setting, we see the possibility to generate a conformance test suite based on the strings in  $UV$ . A conformance test suite based on the W technique is for  $\mathcal{M}$  the set  $U(\varepsilon \cup \Sigma \cup \Sigma^2 \cup \dots \cup \Sigma^{l-k})V$  [8]. If  $\mathcal{A}$  and  $\mathcal{M}$  disagree, at least one counter-example will be found among the set of tests  $U(\varepsilon \cup \Sigma \cup \Sigma^2 \cup \dots \cup \Sigma^{l-k})V$  with the exception of  $UV$ , so there is no need to test these again.



## Chapter 3

# Summary of Papers

This chapter contains summaries of and retrospective discussions about the work in the papers included in the thesis.

### 3.1 Paper A: Learning Finite State Machines

Paper A is included in a book on model-based testing of reactive systems [7]. The paper contains a survey of the most well-known techniques for regular inference. It is intended as a tutorial on the subject of regular inference. The survey contains presentations of Angluin's  $L^*$  algorithm [1], the algorithm using a reduced observation table by Rivest and Schapire [27], and the algorithm using a discrimination tree by Kearns and Vazirani [18]. In the paper, these algorithms are presented using the unifying concept of observation packs, presented by Balcázar et al. [2]. The presentation of the algorithms contains a theoretical background, their complexity, the algorithms presented in pseudo code, and an illustrating example. The survey also suggests two different ways to realize equivalence queries.

The paper also presents an introduction to domain-specific optimizations in regular inference [15, 4]. These optimizations are based on the idea that knowledge about the domain of the system can be used to reduce the number of membership queries required by a regular inference algorithm. Another application of regular inference is also presented, so called adaptive model checking [11]. This is a technique to combine regular inference and model checking. Model checking can be applied to a model, generated by regular inference, even though the model is not yet a correct model of the system. The time to generate a model and perform model checking is reduced if a counter-example to the checked property is found during model checking of a relatively small model. Counter-examples from the model checker may in certain cases also be used as counter-examples in the regular inference algorithm.

Practical results collected from papers on applying Angluin's  $L^*$  algo-

rithm on adaptive model-checking and domain-specific optimizations to examples [11, 15, 4] are presented in short.

### 3.2 Paper B: Insight's to Angluin's Learning

We have implemented the  $L^*$  algorithm for regular inference suggested by Dana Angluin. The implementation is created in a straight-forward manner in Java and uses the library AMoRE [20] for manipulating automata. It is used to analyze the performance of the  $L^*$  algorithm on randomly generated examples and a handful of protocols supplied with the Edinburgh Concurrency Workbench (CWB) [21]. The focus of our analysis is foremost on the number of membership required by the algorithm on different types of examples. Membership queries are counted since they are likely to be the most time-consuming activity when using regular inference in practice.

We have experimented with two types of randomly generated examples. The automata in the first set of examples are randomly generated with respect to transitions and the percentage of accepting states. Let  $n$ ,  $|\Sigma|$ , and  $m$  be as described in Section 2.2, and let  $|\delta| = n|\Sigma|$  be the number of transitions. In our experiments we have used the shortest possible counter-example, hence the longest counter-example  $m$  is at most  $n$ . On our examples the number of membership queries grows approximately linearly with the number of transitions,  $|\delta|$ . This result is in contrast to the upper bound of the algorithm in this setting, which is  $n^2|\delta|$ , i.e. quadratic in the number of states.

The second type of randomly generated examples is constructed in a way to imitate examples of reactive systems. The examples are prefix-closed DFAs. The number of membership queries for these type of examples grows approximately quadratically in the number of transitions,  $|\delta|^2$ . This is closer to the upper bound of the algorithm in comparison to the first type of examples.

We hoped to improve this result with the aid of a domain-specific optimization for prefix-closed systems [4]. The optimization makes use of knowledge about prefix-closed languages, extensions of not accepted strings are not accepted. This payed off in an approximately 20% reduction of membership queries.

We also applied Angluin's algorithm to a small set of examples supplied by CWB and compared the result to the algorithm applied to a set of randomly generated prefix-closed DFAs. We found that on average the CWB examples required 7% fewer membership queries without and 35% with the optimization, in comparison to the randomly generated prefix-closed DFAs. This might indicate that real-world examples exhibit a certain structure which makes the algorithm perform better.

## Discussion

The longer-term goal of our work is to apply regular inference to a system larger than the examples, on which results from applying Angluin’s  $L^*$  algorithm has been reported in literature [15]. Therefore we wanted to investigate how the  $L^*$  algorithm scales with size of systems; in particular, how it performs on prefix-closed DFAs.

Our results for prefix-closed DFAs show that these generally require more membership queries than general DFAs. This result indicates that randomly generated DFAs are not reliable to evaluate how the  $L^*$  algorithm scales for reactive systems. We believe that the cause for this difference in the number of required membership queries is that it is harder to distinguish states from each other in prefix-closed DFAs.

Our implementation is coded in Java and no particular effort was spent to optimize the data structures. Hence we see room for optimizations in the time and memory consumption. Optimizing the implementation would allow us to experiment with larger models than we did. On the other hand, the trend for the different categories of examples were clear and conclusions regarding the growth of membership and equivalence queries could be drawn from the results.

### 3.3 Paper C: Regular Inference for State Machines with Parameters

In our previous work we experienced that models of reactive systems required a larger amount of membership queries compared to randomly generated examples [4]. From this point on, our work shifted towards domain-specific optimizations. We have adopted Angluin’s  $L^*$  algorithm to work more efficiently on a particular class of systems of interest, entities of communication protocols.

A state machine representing an entity of a communication protocol can have a very large alphabet, since a symbol typically consists of a protocol data unit type with a number of parameters, each of which can assume many values. For this particular type of system we developed an optimization to the  $L^*$  algorithm.

The idea of the optimization is to infer, for each state, a partitioning of input symbols into equivalence classes, under the hypothesis that all input symbols in an equivalence class have the same effect on the state machine. Whenever such a hypothesis is disproved, the partitioning of input symbols is refined.

We also formed a second idea to further optimize Angluin’s regular inference algorithm. The idea is to reduce the number of membership queries required to construct a model. We realize this by relaxing the requirement

that every entry in the observation table must be filled.

The two optimizations form our regular inference algorithm for state machines with parameters. We implemented our algorithm in LearnLib [26], a library for regular inference, in order to make a comparison between our and Angluin’s  $L^*$  algorithm.

We focused on generating and running synthesized examples that varied the number of equivalence classes of input symbols. The result from exercising our algorithm on the examples illustrates how the algorithm grows in proportion to the number of parameters affecting the state machine. This is the result we had hoped for; when there is a low proportion of input symbols affecting the system, we can reduce a large amount of membership queries by applying our algorithm. However, the negative result is that the required number of equivalence queries is larger with our algorithm. This can be explained by the fact that with our algorithm, models can be generated based on fewer membership queries, i.e., less information; hence a larger number of incorrect models may be created.

## Discussion

The result presented in this paper is the outcome of the two optimizations combined into a new algorithm. The optimizations can theoretically be applied separately but they are not so in this investigation. It may be of interest to insert this separation of the optimizations into the implementation in order to see with what they each contribute.

We believe that the large number of equivalence queries required by our algorithm contributes to the growth of membership queries. The reason is that Angluin’s algorithm requires that all prefixes of counter-examples must be queried for and also all one symbol extensions of the prefixes. We believe that a more effective way to handle counter-example will reduce the required number of membership queries.

## Chapter 4

# Conclusions and Future Work

We have conducted a survey of the most well-known techniques for regular inference. The survey contains presentations of regular inference algorithms by Angluin [1], by Rivest and Schapire [27], and by Kearns and Vazirani [18]. We also present an introduction to domain-specific optimizations in regular inference [15, 4], optimizations that are based on knowledge about the domain of the system. Another application of regular inference is also presented, a technique to combine regular inference and model checking, called adaptive model checking [11].

We have implemented one of the surveyed algorithms, the  $L^*$  algorithm, in order to investigate how the  $L^*$  algorithm scales with size of systems; in particular, how it performs on examples constructed in a way to imitate examples of reactive systems. These examples are prefix-closed DFAs. Our results for prefix-closed DFAs show that these generally require more membership queries than general DFAs. This result indicates that randomly generated DFAs are not reliable to evaluate how the  $L^*$  algorithm scales for reactive systems. We improve the result for prefix-closed DFAs with the aid of a domain-specific optimization for prefix-closed systems [4]. It paid off by a reduction of approximately 20% of membership queries.

The technique of regular inference can be used to construct models of entities of communication protocols. Typically in such protocols, symbols consists of a message type together with a number of parameters, each of which can assume several values. Hence, the alphabet of such an entity is exponential in the number of parameters and this negatively affects the required amount of membership queries. If only a few of the parameters determine the subsequent behavior of the system it is desirable to avoid asking membership queries for the parameter settings that have no effect. Therefore we developed a regular inference algorithm that specializes in handling these cases [5]. The result of applying the algorithm to a large

set of randomly generated examples shows that the algorithm requires less membership queries compared to  $L^*$  when only a few of the parameters determine the subsequent behavior of the system.

In the future we would like to apply the regular inference for systems with parameters on a realistic communication protocol entity. A suggestion to this is a larger piece of the Network Presence Center product of Mobile Arts AB [6], compared to the fragment that has been investigated [5]. This investigation would clarify how useful the optimizations for systems with parameters are and may lead to other adaptations as well.

A second interesting work is to improve the handling of counter-examples in regular inference. In the current situation there are mainly two ways to handle counter-examples. Either to insert the whole counter-example and possibly all of its prefixes, as in  $L^*$  and the reduced observation table [1, 27], or to extract an interesting suffix as in Kearns' and Vazirani's regular inference algorithm [18]. There is a simple optimization, a binary search in the counter-example, to find a suffix which puts the model in a rejecting state but the system in an accepting state, or vice versa. Only the suffix of the counter-example is then used in the algorithm to update the model. A more optimal handling of counter-examples would improve the performance of regular inference algorithms.

The implementation of the optimizations for systems with parameters [5] is not included in LearnLib [26]. It would be of value to incorporate these optimizations into LearnLib, so that students, researchers, and other interested in using the tool has access to these optimizations as well.

# Bibliography

- [1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [2] J.L. Balcázar, J. Díaz, and R. Gavaldá. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72. Kluwer, 1997.
- [3] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In Maura Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2005.
- [4] Therese Berg, Bengt Jonsson, Martin Leucker, and Mayank Saxena. Insights to angluin’s learning. *Electr. Notes Theor. Comput. Sci.*, 118:3–18, 2005.
- [5] Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines with parameters. In *FASE*, volume 3922 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2006.
- [6] J. Blom and B. Jonsson. Automated test generation for industrial erlang applications. In *Proc. 2003 ACM SIGPLAN workshop on Erlang*, pages 8–14, Uppsala, Sweden, Aug. 2003.
- [7] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
- [8] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. on Software Engineering*, 4(3):178–187, May 1978. Special collection based on COMPSAC.
- [9] J.-C. Fernandez, C. Jard, T. Jérón, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming*, 29, 1997.

- [10] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Trans. on Software Engineering*, 17(6):591–603, June 1991.
- [11] A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In J.-P. Katoen and P. Stevens, editors, *Proc. TACAS '02, 8<sup>th</sup> Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 357–370. Springer Verlag, 2002.
- [12] John Hatcliff, Matthew B. Dwyer, and Hongjun Zheng. Slicing software for model construction. *Higher-Order and Symbolic Computation*, 13(4):315–353, 2000.
- [13] Gerard J. Holzmann and Margaret H. Smith. Software model checking: extracting verification models from source code. *Software Testing, Verification and Reliability*, 11(2):65–79, 2001.
- [14] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, SE-23(5):279–295, May 1997.
- [15] H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *Proc. 15<sup>th</sup> Int. Conf. on Computer Aided Verification*, 2003.
- [16] ITU-T. Recommendation Z.120, Message Sequence Charts. In *Languages and general software aspects for telecommunication systems*, Geneva, Nov. 1999.
- [17] ITU-T. Recommendation Z.100, Specification and Description Language (SDL). In *Languages and general software aspects for telecommunication systems*, Geneva, Aug. 2002.
- [18] M.J. Kearns and U.V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
- [19] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines – a survey. *Proc. IEEE*, 84(8):1090–1126, 1996.
- [20] O. Matz, A. Miller, A. Potthoff, W. Thomas, and E. Valkema. Report on the program amore. Technical Report 9507, Inst. f. Informatik u. Prakt. Math., CAU Kiel, 1995.
- [21] Faron Moller and Perdita Stevens. Edinburgh Concurrency Workbench user manual (version 7.1). Available from <http://homepages.inf.ed.ac.uk/perdita/cwb/>.
- [22] Oliver Niese. An integrated approach to testing complex systems. Technical report, Dortmund University, 2003. Dissertation.

- [23] OMG. *Unified Modeling Language, UML*, Mar 2003.
- [24] D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. In J. Wu, S. T. Chanson, and Q. Gao, editors, *Formal Methods for Protocol Engineering and Distributed Systems, FORTE/PSTV*, pages 225–240, Beijing, China, 1999. Kluwer.
- [25] A. Pnueli. The temporal logic of programs. In *Proc. 18<sup>th</sup> Annual Symp. Foundations of Computer Science*, pages 46–57. IEEE, 31 October–2 November 1977.
- [26] H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 62–71, New York, NY, USA, 2005. ACM Press.
- [27] R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.
- [28] M. P. Vasilevski. Failure diagnosis of automata. *Cybernetic*, 9(4):653–665, 1973.
- [29] Willem Visser, Klaus Havelund, Guillaume P. Brat, Seungjoon Park, and Flavio Lerda. Model checking programs. *Autom. Softw. Eng.*, 10(2):203–232, 2003.
- [30] Mark Weiser. Program slicing. In *ICSE '81: Proceedings of the 5th international conference on Software engineering*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.





## Recent licentiate theses from the Department of Information Technology

- 2006-003**    Therese Berg: *Regular Inference for Reactive Systems*
- 2006-002**    Anders Hessel: *Model-Based Test Case Selection and Generation for Real-Time Systems*
- 2006-001**    Linda Brus: *Recursive Black-box Identification of Nonlinear State-space ODE Models*
- 2005-011**    Björn Holmberg: *Towards Markerless Analysis of Human Motion*
- 2005-010**    Paul Sjöberg: *Numerical Solution of the Fokker-Planck Approximation of the Chemical Master Equation*
- 2005-009**    Magnus Evestedt: *Parameter and State Estimation using Audio and Video Signals*
- 2005-008**    Niklas Johansson: *Usable IT Systems for Mobile Work*
- 2005-007**    Mei Hong: *On Two Methods for Identifying Dynamic Errors-in-Variables Systems*
- 2005-006**    Erik Bängtsson: *Robust Preconditioned Iterative Solution Methods for Large-Scale Nonsymmetric Problems*
- 2005-005**    Peter Naucér: *Modeling and Control of Vibration in Mechanical Structures*
- 2005-004**    Oskar Wibling: *Ad Hoc Routing Protocol Validation*
- 2005-003**    Magnus Ågren: *High-Level Modelling and Local Search*
- 2005-002**    Håkan Zeffer: *Hardware-Software Tradeoffs in Shared-Memory Implementations*
- 2005-001**    Jesper Wilhelmsson: *Efficient Memory Management for Message-Passing Concurrency — part I: Single-threaded execution*



UPPSALA  
UNIVERSITET