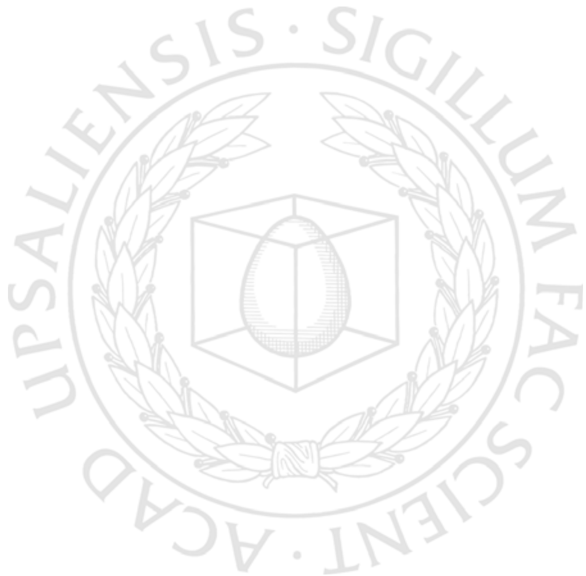
UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 434

Learning of Timed Systems

OLGA GRINCHTEIN



ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2008

ISSN 1651-6214
ISBN 978-91-554-7207-8
urn:nbn:se:uu:diva-8763

Dissertation presented at Uppsala University to be publicly examined in polhemsalen, Ångström Laboratory, Uppsala University, Uppsala, Friday, May 23, 2008 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract
Grinchtein, O. 2008. Learning of Timed Systems. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 434. 44 pp. Uppsala. ISBN 978-91-554-7207-8.

Regular inference is a research direction in machine learning. The goal of regular inference is to construct a representation of a regular language in the form of deterministic finite automaton (DFA) based on the set of positive and negative examples. DFAs take strings of symbols (words) as input, and produce a binary classification as output, indicating whether the word belongs to the language or not. There are two types of learning algorithms for DFAs: passive and active learning algorithms. In passive learning, the set of positive and negative examples is given and not chosen by inference algorithm. In contrast, in active learning, the learning algorithm chooses examples from which a model is constructed.

Active learning was introduced in 1987 by Dana Angluin. She presented the L* algorithm for learning DFAs by asking membership and equivalence queries to a teacher who knows the regular language accepted by DFA to be learned. A membership query checks whether a word belongs to the language or not. An equivalence query checks whether a hypothesized model is equivalent to the DFA to be learned.The L* algorithm has been found to be useful in different areas, including black box checking, compositional verification and integration testing. There are also other algorithms similar to L* for regular inference. However, the learning of timed systems has not been studied before. This thesis presents algorithms for learning timed systems in an active learning framework.

As a model of timed system we choose event-recording automata (ERAs), a determinizable subclass of the widely used timed automata. The advantages of ERA in comparison with timed automata, is that it is known priori the set of clocks of an ERA and when clocks are reset. The contribution of this thesis is four algorithms for learning deterministic event-recording automaton (DERA). Two algorithms learn a subclass of DERA, called event-deterministic ERA (EDERA) and two algorithms learn general DERA.

The problem with DERAs that they do not have canonical form. Therefore we focus on subclass of DERAs that have canonical representation, EDERA, and apply the L* algorithm to learn EDERAs. The L* algorithm in timed setting requires a procedure that learns clock guards of DERAs. This approach constructs EDERAs which are exponentially bigger than automaton to be learned. Another procedure can be used to lean smaller EDERAs, but it requires to solve NP-hard problem.

We also use the L* algorithm to learn general DERA. One drawback of this approach that inferred DERAs have a form of region graph and there is blow-up in the number of transitions. Therefore we introduce an algorithm for learning DERA which uses a new data structure for organising results of queries, called a timed decision tree, and avoids region graph construction. Theoretically this algorithm can construct bigger DERA than the L* algorithm, but in the average case we expect better performance.

Keywords: learning regular languages, timed systems, event-recording automata

Olga Grinchtein, Department of Information Technology, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

© Olga Grinchtein 2008

ISSN 1651-6214
ISBN 978-91-554-7207-8
urn:nbn:se:uu:diva-8763 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-8763)

List of Papers

This thesis is based on the following papers, which are referred to in the text by the capital letters A and B.

A. **Learning of Event-Recording Automata.** Olga Grinchtein, Bengt Jonsson, Martin Leucker. *Technical report 2008-013, Uppsala University 2008.* Revised version of two papers

- **Learning of Event-Recording Automata.** Olga Grinchtein, Bengt Jonsson, Martin Leucker. *In Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault-Tolerant Systems, volume 3253 of LNCS, pages 379-396. Springer 2004.*
- **Inference of Timed Transition Systems.** Olga Grinchtein, Bengt Jonsson, Martin Leucker. *In Proceedings of International Workshop on Verification of Infinite State Systems, Electronic Notes in Theoretical Computer Science, 138(3):87-99, 2005.*

B. **Inference of Event-Recording Automata using Timed Decision Trees.** Olga Grinchtein, Bengt Jonsson. *Technical report 2008-014, Uppsala University 2008.* Revised version of the paper **Inference of Event-Recording Automata using Timed Decision Trees.** Olga Grinchtein, Bengt Jonsson, Paul Pettersson. *In Proceedings of International Conference on Concurrency Theory, volume 4137 of LNCS, pages 435-449. Springer 2006.*

Comments on my Participation

A. I participated in the discussions and in the writing of the paper.

B. I participated in the discussions and in the writing of the paper.

Other Publications

- Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the Correspondence Between Conformance Testing and Regular Inference. *In Proceedings of International Conference on Fundamental Approaches to Software Engineering , volume 3442 of LNCS, pages 175-189. Springer 2005.*

- Olga Grinchtein, Martin Leucker and Nir Piterman. Inferring Network Invariants Automatically. *In Proceedings of International Conference on Automated Reasoning, volume 4130 of LNCS, pages 483-497. Springer 2006.*

- Olga Grinchtein and Martin Leucker. Learning Finite-State Machines from Inexperienced Teachers. *In Proceedings of International Colloquium on Grammatical Inference: Algorithms and Applications, volume 4201 of LNCS, pages 344-345. Springer 2006.*

- Johannes Borgström, Olga Grinchtein and Simon Kramer. Timed Calculus of Cryptographic Communication. *In Proceedings of International Workshop on Formal Aspects in Security and Trust, volume 4691 of LNCS, pages 16-30. Springer 2006.*

- Olga Grinchtein and Martin Leucker. Network Invariants for Real-Time Systems. *Formal Aspects of Computing, 2008, accepted for publication.*

Summary in Swedish

Maskininlärning är ett område inom datavetenskap som innehåller ett vitt spektrum tekniker: begreppsinlärning, beslutsträdskonstruktion, genetiska algoritmer, belöningsbaserad inlärning, för att nämna några. Metoder för maskininlärning har använts på problem inom bl.a. molekylärbiologi, taligenkänning, reglerteknik och statistik. Till exempel har maskinlärning använts för att förutsäga proteiners sekundärstruktur, för att dela upp mänskligt tal i meningar, och för att klassificera data av olika slag.

Vi är intresserade av en inriktning inom maskininlärning som kallas begreppsinlärning. Ett begrepp kan här ses som en regel som klassificerar objekt inom en mängd i de objekt som faller inom begreppet (positiva exempel) och de som inte faller inom begreppet (negativa exempel). Målet med begreppsinlärning är att härleda ("lära in") en klassificeringsregel som beskriver ett okänt begrepp utifrån ett antal positiva och negativa exempel.

Inom begreppsinlärning finns två inriktningar, passiv inlärning och aktiv inlärning. I passiv inlärning är mängden positiva och negativa exempel givna och inte valda. Detta till skillnad från aktiv inlärning, där inlärningsalgoritmen kan välja exempel från vilka ett begrepp lärs.

För att illustrera aktiv inlärning, anta att en förare vill ta reda på när ett visst trafikljus byter mellan grönt och rött, med andra ord hur länge trafikljuset är grönt och hur länge trafikljuset är rött. Föraren närmar sig trafikljuset när det är rött. Efter 20 sekunder slår trafikljuset om till grönt och föraren korsar vägen efter ytterligare 20 sekunder, eftersom det finns bilar framför honom. Föraren kan då dra slutsatsen att trafikljuset är rött minst 20 sekunder. Nästa gång föraren närmar sig korsningen måste han vänta 25 sekunder tills det röda ljuset blir grönt och efter ytterligare 10 sekunder kan han korsa vägen. Nu kan föraren dra slutsatsen att trafikljuset är rött i minst 25 sekunder och grönt i minst 20 sekunder. Baserat på dessa observationer kan han modellera trafikljusets beteende som i figur 1.

Föraren diskuterar trafikljusets beteende med en kollega på arbetet. Kollegan har kört samma väg många gånger, och kan berätta att trafikljuset är rött i minst 30 sekunder. Då bör modellen ändras igen, till en ny modell som i figur 2.

Således, varje gång föraren korsar vägen får han nya positiva och negativa exempel på trafikjusets beteende. Vägledd av dessa konstruerar han en hypotes (en modell av trafikljusets beteede), ber sedan sin kollega om ett motexempel (dvs. ett beteende som inte stämmer med hypotesen), och förfinar därefter modellen.

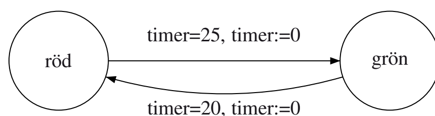


Figure 1: Trafikljuskontroller. I den här bilden indikerar cirklar tillstånd hos trafikljuskontrollern. Cirklar som är markerade i rött (grönt) representerar tillstånd när ljuset är rött (grönt). Pilar representerar transitioner mellan tillstånd. Efter 25 sekunder växlar rött ljus till grönt, trafikljuskontroller förflyttar sig från det röda tillståndet till det gröna tillståndet och timern ställs om. Efter 20 sekunder växlas det gröna ljuset till rött, trafikljuskontroller förflyttar sig ifrån det gröna tillståndet till det röda tillståndet och timern nollställs.

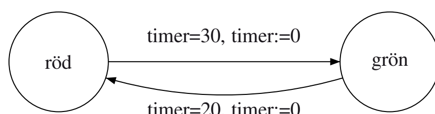


Figure 2: Trafikljuskontroller. Efter 30 sekunder växlar det röda trafikljuset till grönt och timern nollställs. Efter 20 sekunder växlas det gröna ljuset till rött, trafikljuskontroller förflyttar sig ifrån det gröna tillståndet till det röda tillståndet och timern nollställs.

Automater, som de i ovanstående figurer, kan användas för att beskriva hur en viss mängd händelser får ordnas efter varandra i tiden. De används för många tillämpningar, t.ex. kommunikationsprotokoll (i vilken ordning får meddelanden skickas och tas emot?) och styrsystem (i vilken ordning ska olika operationer utföras?). En automat kan således ses som ett begrepp som kan läras in från exempel på tillåtna och otillåtna sekvenser av händelser.

Aktiv inlärning introducerades av Dana Angluin 1987. Hon presenterade en algoritm som lär sig deterministiska ändliga automater. Därefter har många andra algoritmer för liknande begrepp utvecklats. Emellertid har man inte utvecklat algoritmer för inlärning av automater som mäter avståndet i tid mellan olika händelser, och låter dessa avstånd styra vilka sekvenser som är tillåtna eller inte. Sådana automater kan avändas för att beskriva så kallade tidsberoende system, som skiljer sig från system utan tid genom att deras beteende inte enbart kan beskrivas genom vilka händelseföljder som är tillåtna, utan också beror på tiden när en händelse inträffar. Ett trafikljus är ett exempel på ett tidsberoende system. En populär modell för tidsberoende system är tidsautomater. En tidsautomat använder klockor för att modellera beteenden som beror av tid. I figur 1 och figur 2 visas två exempel på tidsautomater med klocka som kallas "timer".

Målet med avhandlingen är att utveckla aktiva inlärningalgoritmer som härleder modeller av tidssystem. Huvudproblemet är att modellera tidsbeteende baserat på positiva och negativa exempel. I figur 1 visas en enkel modell, i vilken trafikljuset kan förflytta sig ifrån ett tillstånd till ett annat tillstånd endast vid en viss tidpunkt. I allmänhet kan en timer anta värden från något intervall. I detta fall behöver vi härleda intervall från en mängd exempel, vilket inte alltid är lätt.

I den här avhandlingen visar vi att för några klasser av tidsberoende system kan Angluins algoritm användas, men den måste utvidgas med procedurer för att härleda hur klockorna påverkar beteendet. Nackdelen med vår utvidgning av Angluins algoritm är att den konstruerar modeller av tidsberoende system med ett mycket stort antal tillstånd och transitioner. Därför utvecklar vi en annan algoritm som försöker lösa dessa problem.

Acknowledgments

I owe a great debt of gratitude to my supervisor Bengt Jonsson for enormous help which I have from you, for encouraging me to think, for long discussions, for being patient with me and for teaching me how to do research. It was my great fortune to be your student.

I am indebted to Martin Leucker for an exciting and very productive collaboration, for supporting my projects, for the enthusiasm to work with me, and for being my host during my visits to the Technische Universität München.

My special thanks to Rafał Somla for his very kind help with the courses I taught and for overall support during all years of my PHD.

I would like to thank members of research group “Testing of Reactive Systems”, i.e. Therese Berg, Johan Blom, Anders Hessel and Paul Pettersson, for interesting discussions. I am especially grateful to Paul Pettersson for helping with the preparation of the CONCUR’06 paper and to Therese Berg, who graciously offered her aid with the translation of the summary into Swedish.

I am grateful to Nir Piterman for fruitful discussions during my visit to Ecole Polytechnique Fédérale de Lausanne and for helping me with the IJCAR’06 paper.

I am grateful to Simon Kramer, Pavel Krčál and Justin Pearson for reading parts of the thesis and providing valuable comments.

I am thankful to all people at the department for creating friendly atmosphere. It was a great pleasure to work with all of you!

Last, but not least, I wish to warmly thank my mother and my brother for their support and encouragement.

Contents

1 Introduction . . . 15
2 Learning of Deterministic Finite Automata . . . 21
 2.1 Angluin's algorithm . . . 22
 2.2 Trakhtenbrot-Barzdin's algorithm . . . 25
3 Timed systems . . . 27
 3.1 Timed and Event-Recording Automata . . . 27
 3.2 Problems with DERA . . . 29
4 Contribution . . . 31
 4.1 Paper A . . . 31
 4.2 Paper B . . . 32
5 Related Work . . . 35
6 Conclusion and Future Work . . . 37
 6.1 Conclusion . . . 37
 6.2 Future work . . . 38
Bibliography . . . 41

1. Introduction

According to the American Heritage Dictionary of the English language, *learning* is "the act, process, or experience of gaining knowledge or skill". *Machine learning* is a field in computer science that contains a wide spectrum of techniques: reinforcement learning, concept learning, decision tree learning, instance-based learning, and many other techniques. Machine learning methods have been applied to problems in structural molecular biology, speech recognition, control theory, statistics, and many other areas.

We are interested in a branch of machine learning which is called *concept learning*. A *concept* is a classification rule that partitions a domain of instances into those instances that satisfy the rule and those that do not [Utg86]. The goal of concept learning is to infer a classification rule that describes an unknown concept. Let us say we want to learn the concept of a family car [Alp04]. The goal is to learn a description of a family car such that given a car which we have not seen before, by checking with this description, we will be able to say whether or not it is a family car. Assume that the features that separate a family car from other cars are the price and engine power. We have a set of examples of cars, and we have a group of people to whom we show these cars. The people look at the cars we show them and label them: either family car or not family car. The cars that they consider to be family cars are positive examples and the other cars are negative examples. The concept learning algorithm generates a description of a family car that is shared by all positive examples and none of the negative examples. The resulting description can e.g. be a formula, where a family car's price and engine power should be in a certain range, for example, the price is between $150,000$ and $250,000$ kronor, and the engine power is between 160 and 200 hp.

In the field of concept learning there are two main strands, *passive learning* and *active learning*. In passive learning, the set of positive and negative examples is given and not chosen. In contrast, in active learning, the learning algorithm chooses examples from which a concept is learned.

Let us consider active learning in more detail. Active learning has two participants: a $Learner$ and a $Teacher$. The $Teacher$ knows a concept to learn. The goal of the $Learner$ is to learn the concept by asking queries to the $Teacher$. In the family car example, the Teacher is the set of people that label cars. There are two types of queries: membership queries and equivalence queries. A membership query can be for example whether a certain car is a family car, to which the $Teacher$ replies yes or no. Based on answers to membership queries the

Learner can construct a *hypothesis*, for example, a price for a family car is between 200000 and 250000 kronor, and an engine power is between 160 and 200 hp, and ask the *Teacher* an equivalence query, whether the hypothesis is correct. The *Teacher* can reply with *yes* or provide a counterexample, for example, a car whose price is 150000 kronor.

Different concept learning algorithms have been developed for different types of concepts. In this thesis we study learning of computational structures. Examples of computational structures are computable functions, programs, deterministic finite automata, and grammars. The learning of *deterministic finite automata* (DFAs), also called *regular inference*, is the research direction, which is most related to this thesis. DFAs accept regular languages and can be used to model the behavior of reactive systems, such as sequential circuits, communication protocols and embedded controllers. There are applications of DFA in natural language processing, encryption algorithms, operating system analysis, etc.

There are many algorithms for regular inference [Ang87], [RS93], [Mur96], [BDG97], [PNH98], [PH01], [BO05]. An early work by Gold [Gol78] proved that the problem of finding the smallest DFA consistent with a set of positive and negative examples (passive learning), is NP-hard. Later Angluin [Ang87] proved that if we consider active learning instead of passive learning, then DFAs are learnable after asking a polynomial number of membership and equivalence queries. Algorithms for regular inference have been applied to detecting malicious web requests [ISBF07], to identifying the protein α-chain region in amido acid sequences [YIK94], to learn topological maps used to navigate a robot [Kun01], to generate models of component behavior in compositional verification [CGP03], and to classify natural language sentences [LGF00].

Let us consider an example of regular inference. A classical example of a reactive system is an automatic controller that opens and closes gates at a railroad crossing. The system consists of three components: a train, a gate and a controller. There are two sensors: the first one detects the approach of the train towards the gate and the second one detects its exit. The command given by the controller to the gate depends on the signal received from the sensors. Whenever the controller receives the signal *approach* from the sensor, it responds by sending the command *lower* to the gate. Whenever it receives the signal *exit*, it responds with the command *raise* to the gate. Based on this description of the controller we construct the following set of words, and label each as either positive or negative example. In the below table we use "+" for

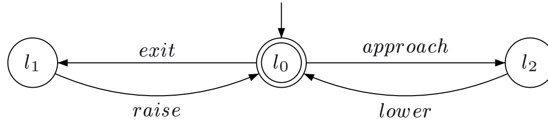


Figure 1.1: A DFA for a controller

positive example and “-” for negative example.

$$\begin{array}{ll}
(approach) & - \\
(approach)(lower) & + \\
(exit) & - \\
(exit)(raise) & + \\
(approach)(raise) & - \\
(exit)(lower) & -
\end{array}$$

The goal of regular inference is to construct a DFA, which models the behavior of the controller, from this set of examples. An example of a DFA that can be constructed by regular inference from this set is shown in Figure 1.1.

The set of examples for the controller models the sequencing of signals, but not the actual times at which the signals occur. The response time of the controller to the $approach$ signal can be 1 second, and to the signal $exit$ at most 1 second. Timing can be added to the words, by coupling every signal with the time at which it occurs (in seconds). Then the following set of *timed words* can be obtained.

$$\begin{array}{ll}
(approach, 1) & - \\
(approach, 1)(lower, 1.5) & - \\
(approach, 1)(lower, 2) & + \\
(approach, 1)(lower, 2.5) & - \\
(exit, 1) & - \\
(exit, 1)(raise, 1) & + \\
(exit, 1)(raise, 1.5) & + \\
(exit, 1)(raise, 2) & - \\
(approach, 1)(raise, 1) & - \\
(exit, 1)(lower, 1) & -
\end{array}$$

Our goal is to infer a timed model of the controller based on this set of examples. One modelling formalism for timed systems is *timed automata* [AD94]. A timed automaton is a finite automaton with clocks, which model timed behavior. A *timed automaton* for the controller is shown in Figure 1.2. The timed

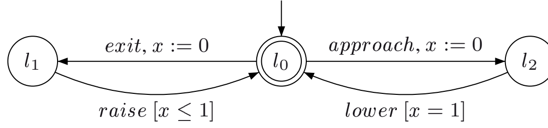


Figure 1.2: A timed automaton for a controller

automaton contains a clock x. An annotation of the form $x := 0$ on an edge means that the clock x is reset when the edge is traversed. An annotation of the form $x \leq 1$ on the edge from l_1 to l_0 means that transition from l_1 to l_0 is enabled if the value of clock x is at most 1.

The interesting question is how to infer timed automata. This problem has not been studied very much. One of the few papers is by Verwer et. al [VdWW06] who present an algorithm that infers timed automata with one clock. Besides the limitation on the number of clocks, it is a passive learning algorithm which uses a trial and error strategy. Thus, it would be interesting to develop an active learning algorithm that infers timed automata in a systematic way. The goal of this thesis is *to develop active learning algorithms that infer models of timed systems in the form of timed automata*.

To illustrate active learning we present three examples of applications of regular inference. These occur in black box checking, compositional verification, and integration testing.

Two main approaches for increasing the quality of systems are model checking, in which one checks whether a given abstract design satisfies some given property, and conformance testing, in which one checks whether a given implementation, whose internal structure is often unknown, conforms with an abstract design. Peled et. al [PVY99] present a black box checking algorithm which combines these techniques, i.e., tests whether an implementation with unknown internal structure (a black box) satisfies some given property. The algorithm learns a model of the internal structure of the implementation using regular inference and performs model checking to check whether the constructed model satisfies the given property. If the model satisfies the property, then conformance testing is performed. If a counterexample is obtained, i.e behaviour of the model that violates the property, then it is checked whether the counterexample is a run of the implementation. If the counterexample is not a run of the implementation, then regular inference refines the model, otherwise the implementation does not satisfy the given property.

In verification of large software systems, the system is often decomposed into small parts so that each component can be separately verified. This method is called compositional verification. To verify that a system M composed of two modules M_1 and M_2 satisfies a property φ, one possible approach is to find an abstraction A of M_2 such that the composition of M_1 and A satisfies φ

and A is significantly smaller than M_2. The task of finding A is typically left to the user. However, if the module A can be represented as a DFA then A can be constructed using active learning [CGP03], [AMN05], [Leu07].

Li et al. [LGS06] use active learning for integration testing. In integration testing, software or hardware components are combined and tested to evaluate the interaction between them. When components come from third party sources, their internal structure is unknown. Regular inference can then be used to construct a model for each third party component. Thereafter models of components are integrated and the whole system is tested based on the models of the components according to a certain test generation strategy.

The rest of this introduction is structured as follows. In the next section we introduce regular inference by presenting two representative algorithms for regular inference. In Section 3 we present timed automata and discuss some problems encountered when trying to extend regular inference to timed systems. In Section 4 we present the contribution of the thesis. In Section 5 we present related work, and Section 6 contains a discussion of future work.

2. Learning of Deterministic Finite Automata

One of the successful fields in concept learning is regular inference, i.e. the problem of constructing a deterministic finite automaton from information about the set of words it accepts. Many regular inference algorithms were introduced since the 70s. They can be distinguished by the data structure which they use and the type of learning which they employ (passive or active). In 1973 Trakhtenbrot and Barzdin presented a passive learning algorithm for DFAs [TB73]. In 1978 Gold proved that passive learning of DFA is NP-hard problem [Gol78]. In 1987 Angluin introduced the L^* algorithm which performs active learning of DFAs [Ang87]. Thereafter variations of L^* algorithm were introduced [RS93], [BDG97]. Algorithms for learning DFAs use trees [Mur96], observation tables [Ang87], observation packs [BDG97] and reduced observation tables [RS93] as data structures for organizing positive and negative examples.

A *Deterministic Finite Automata* (DFA) is a tuple $\mathcal{A} = \langle \Sigma, L, l_0, \delta, L^f \rangle$, where Σ is the alphabet, L is the set of states, $l_0 \in L$ is the initial state, $\delta : L \times \Sigma \to L$ is the transition function, and $L^f \subseteq L$ is the set of final states.

For a word $w \in \Sigma^*$ we define $\delta(l_0, wa) = \delta(\delta(l_0, w), a)$. We say that the word w is accepted if $\delta(l_0, w) \in L^f$, and rejected otherwise.

In Figure 1.1 is shown an example of a DFA. The alphabet $\Sigma = \{exit, approach, raise, lower\}$ and the DFA contains 3 states. The state l_0 is the initial and final state. The transition function states, e.g. that if in the state l_0 the controller receives signal $approach$, then the controller moves to state l_2, which is defined as $\delta(l_0, approach) = l_2$.

The set of words accepted by DFA $\mathcal{A}$ is called the language accepted by $\mathcal{A}$, and is denoted $\mathcal{L}(\mathcal{A})$. For example, $(approach)(lower)$ is a word accepted by the DFA in Figure 1.1.

An important property of DFAs, which is the basis for regular inference is that DFAs can be characterized in terms of Nerode's right congruence. Let us recall the notion of Nerode's right congruence. Given a language $\mathcal{L}(\mathcal{A})$, we say that two words $u, v \in \Sigma^*$ are *equivalent*, written as $u \equiv_{\mathcal{L}(\mathcal{A})} v$, if, for all $w \in \Sigma^*$, we have $uw \in \mathcal{L}(\mathcal{A})$ iff $vw \in \mathcal{L}(\mathcal{A})$. It is easy to see that $\equiv_{\mathcal{L}(\mathcal{A})} \subseteq \Sigma^* \times \Sigma^*$ is a right congruence, i.e., it is an equivalence relation that additionally satisfies $u \equiv_{\mathcal{L}(\mathcal{A})} v$ implies $uw \equiv_{\mathcal{L}(\mathcal{A})} vw$ for all $w \in \Sigma^*$. We

denote the equivalence class of a word w wrt. $\equiv_{\mathcal{L}(\mathcal{A})}$ by $[w]_{\mathcal{L}(\mathcal{A})}$, or just $[w]$ if $\mathcal{L}(\mathcal{A})$ is clear from the context.

It is folklore that a language $\mathcal{L}(\mathcal{A})$ is regular iff $\equiv_{\mathcal{L}(\mathcal{A})}$ has finite *index*, i.e., the number of equivalence classes of Σ^* with respect to $\equiv_{\mathcal{L}(\mathcal{A})}$ is finite. Let us recall the idea of the proof for the direction right-to-left:

Given a language $\mathcal{L}(\mathcal{A})$ such that $\equiv_{\mathcal{L}(\mathcal{A})}$ has finite index, we construct an automaton $\mathcal{A}_L$ such that $\mathcal{L}(\mathcal{A}_L) = \mathcal{L}(\mathcal{A})$. The states of $\mathcal{A}_L$ are the equivalence classes of Σ^* with respect to $\equiv_{\mathcal{L}(\mathcal{A})}$, the initial state is the equivalence class containing the empty string λ, final states are the ones containing strings in $\mathcal{L}(\mathcal{A})$, and the transition function that for the state $[w]$ and the symbol a prescribes the target state $[wa]$.

It can be shown that this construction yields a minimal (canonical) DFA accepting $\mathcal{L}(\mathcal{A})$, i.e., the number of states is minimal among all DFA accepting $\mathcal{L}(\mathcal{A})$. Furthermore, it can be shown that every minimal DFA is isomorphic to the one we constructed.

In Section 2.1 and Section 2.2 we give an overview of two algorithms which in our opinion are most related to the algorithms presented in the thesis. In Section 2.1 we describe the L^* algorithm due to Angluin and in Section 2.2 we present Traktenbrot-Barzdin's algorithm.

2.1 Angluin's algorithm

The L^* algorithm was introduced by Angluin [Ang87], as an algorithm for learning a DFA from queries. In Angluin's framework there is a $Learner$, who initially knows nothing about $\mathcal{A}$, and a $Teacher$, who has in mind a regular language $\mathcal{L}(\mathcal{A})$ over a known alphabet Σ. The goal of the $Learner$ is to learn $\mathcal{L}(\mathcal{A})$, represented by a minimal DFA, by asking queries to the $Teacher$. The $Learner$ can ask two types of queries

- A *membership query* consists in asking the $Teacher$ whether a word w is in $\mathcal{L}(\mathcal{A})$. The $Teacher$ answers yes or no.

- An *equivalence query* consists in asking the $Teacher$ whether a hypothesized DFA $\mathcal{H}$, which the $Learner$ produces, accept the same language as $\mathcal{A}$, that is whether $\mathcal{L}(\mathcal{A}) = L(\mathcal{H})$. The $Teacher$ answers yes if $\mathcal{L}(\mathcal{A}) = L(\mathcal{H})$, or else supplies a counterexample, i.e. a word w such that $w \in \mathcal{L}(\mathcal{A}) \setminus L(\mathcal{H})$ or $w \in L(\mathcal{H}) \setminus \mathcal{L}(\mathcal{A})$.

The performed membership and equivalence queries constitute a finite collection of words, some of them are members of $\mathcal{L}(\mathcal{A})$ and some of them not. To organize this information, Angluin introduced an *observation table*, which is a tuple (S, E, T) consisting of nonempty finite prefix-closed set S of words, a nonempty finite suffix-closed set E of words, and a finite function T mapping $((S \cup (S \cdot \Sigma)) \cdot E)$ to $\{+, -\}$, where $\cdot$ denotes the concatenation of words.

An observation table can be viewed as a table with rows labelled by elements of $S \cup S \cdot \Sigma$ and columns labelled by elements of E, with the entry for row w and column e equal to $T(w \cdot e)$. The interpretation of T is that for $w \in ((S \cup (S \cdot \Sigma)) \cdot E)$ we have $T(w) = +$ if $w \in \mathcal{L}(\mathcal{A})$, and $T(w) = -$ if $w \notin \mathcal{L}(\mathcal{A})$. If $w \in (S \cup S \cdot \Sigma)$, then $row(w)$ denotes the finite function from E to $\{+, -\}$ defined by $row(w)(e) = T(w \cdot e)$.

We say that an observation table is

- *closed* if for each $w' \in S \cdot \Sigma$ there exists a word $w \in S$ such that $row(w') = row(w)$, and
- *consistent*, if whenever $w_1, w_2 \in S$ are such that $row(w_1) = row(w_2)$, then for all $a \in \Sigma$ we have $row(w_1 \cdot a) = row(w_2 \cdot a)$

In the L^* algorithm the $Learner$ starts by asking membership queries for the empty word λ and each $a \in \Sigma$. Then the $Learner$ constructs the initial observation table (S, E, T), where $S = E = \{\lambda\}$. Given an observation table (S, E, T) it is checked whether (S, E, T) is closed and consistent.

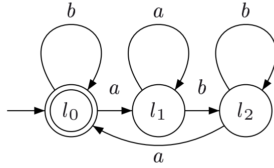
If (S, E, T) is not closed, then the $Learner$ finds $w_1 \in S$ and $a \in \Sigma$ such that $row(w_1 \cdot a) \neq row(w)$ for all $w \in S$. Then the $Learner$ add $w_1 \cdot a$ to S and asks membership queries for all words of the form $w_1 \cdot a \cdot b \cdot e$, where $e \in E$ and $b \in \Sigma$.

If (S, E, T) is not consistent then the $Learner$ finds $w_1, w_2 \in S$, $e \in E$ and $a \in \Sigma$ such that $row(w_1) = row(w_2)$ but $T(w_1 \cdot a \cdot e) \neq T(w_2 \cdot a \cdot e)$. Then the $Learner$ adds the string $a \cdot e$ to E and asks membership queries for all words of the form $w \cdot a \cdot e$, where $w \in S \cup S \cdot \Sigma$.

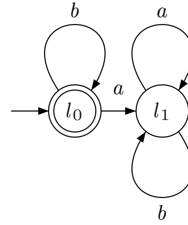
When (S, E, T) becomes closed and consistent, the $Learner$ constructs the hypothesized automaton $\mathcal{H} = \langle \Sigma, L, l_0, \delta, L^f \rangle$, where

- $L = \{row(w) : w \in S\}$,
- $l_0 = row(\lambda)$,
- $\delta(row(w), a) = row(w \cdot a)$,
- $L^f = \{row(w) : w \in S \text{ and } T(w) = +\}$

Based on Nerode's right congruence, two rows $row(w)$ and $row(w')$ for $w, w' \in S \cup S \cdot \Sigma$ such that $row(w) = row(w')$ can be understood as one state in $\mathcal{H}$. Closedness of the observation table (S, E, T) guarantees that the transition function of $\mathcal{H}$ is defined. Consistency of the observation table (S, E, T) guarantees that $\mathcal{H}$ *agrees* with (S, E, T), i.e., that a word $w \in (S \cup S \cdot \Sigma) \cdot E$ is accepted by $\mathcal{H}$ if $T(w) = +$ and rejected by $\mathcal{H}$ if $T(w) = -$. If the observation table (S, E, T) is closed and consistent then the constructed hypothesized DFA $\mathcal{H}$ is the smallest automaton that agrees with (S, E, T). Then the $Learner$ asks an equivalence query to the $Teacher$. The $Teacher$ replies either with yes or provides a counterexample w. If the



(a) An automaton $\mathcal{A}$



(b) An automaton $\mathcal{H}$

Figure 2.1: An automaton to be learned and a hypothesized automaton

T_1	λ
λ	+
a	-
b	+

(a) Table T_1

T_2	λ
λ	+
a	-
b	+
aa	-
ab	-

(b) Table T_2

T_3	λ
λ	+
a	-
ab	-
aba	+
b	+
aa	-
abb	-
$abaa$	-
$abab$	+

(c) Table T_3

T_4	λ	a
λ	+	-
a	-	-
ab	-	+
aba	+	-
b	+	-
aa	-	-
abb	-	+
$abaa$	-	-
$abab$	+	-

(d) Table T_4

Figure 2.2: Observation tables

$Teacher$ replies with yes, then the algorithm terminates with output $\mathcal{H}$. If the $Teacher$ replies with the counterexample w, then the $Learner$ adds w and all its prefixes w' to S and asks membership queries for all words of the form $w \cdot e$, $w \cdot a \cdot e$, $w' \cdot e$, and $w' \cdot a \cdot e$, where $e \in E$ and $a \in \Sigma$.

The L^* algorithm asks at most n equivalence queries and $O(|\Sigma|n^2m)$ membership queries, where n is the number of states in $\mathcal{A}$ and m is the length of the longest counterexample given by the $Teacher$ [Ang87].

Let us consider an example of the L^* algorithm. Let $\mathcal{A}$ be the DFA shown in Figure 2.1(a). Initially, the $Learner$ asks membership queries for λ, a and b. The initial observation table T_1 is shown in Figure 2.2(a), where $S = E = \{\lambda\}$. This observation table is consistent, but not closed, since $row(a) \neq row(\lambda)$. The $Learner$ moves the prefix a to S and then asks membership queries for aa and ab to construct the observation table T_2 shown in Figure 2.2(b). This observation table is closed and consistent. The $Learner$ constructs the hypothesized automaton $\mathcal{H}$ shown in Figure 2.1(b) and asks an equivalence query to the $Teacher$. Assume that the $Teacher$ replies with the counterexample aba, which is in $\mathcal{L}(\mathcal{A})$ but not accepted by $\mathcal{H}$. To process the counterexample aba, the $Learner$ adds ab and aba to S and asks mem-

bership queries for abb, $abaa$ and $abab$ to construct the observation table T_3 shown in Figure 2.2(c). This observation table is closed but not consistent since $row(a) = row(ab)$ but $row(aa) \neq row(aba)$. Then the $Learner$ adds a to E and asks membership queries for ba, aaa, $abba$, $abaaa$ and $ababa$ to construct the observation table T_4 shown in Figure 2.2(d). This observation table is closed and consistent. The $Learner$ constructs the automaton shown in Figure 2.1(a) and asks an equivalence query to the $Teacher$. The $Teacher$ replies yes and L^* terminates.

2.2 Trakhtenbrot-Barzdin's algorithm

In contrast to the L^* algorithm, Trakhtenbrot-Barzdin's algorithm [TB73] is a passive learning algorithm. It constructs a DFA $\mathcal{A}$ from a given sample (S_+, S_-) where S_+ is a set of words accepted by $\mathcal{A}$ and S_- is a set of words rejected by $\mathcal{A}$.

A sample (S_+, S_-) is *m-complete* if it contains every word from Σ^* of length at most m. In Figure 2.3 is shown a 4-complete sample, where $\Sigma = \{a, b\}$, organized as a tree. Every path in the tree corresponds to some word in the sample, and a node has a double circle if and only if the path which leads to the node corresponds to a word accepted by $\mathcal{A}$.

Let $S = S_+ \cup S_-$. We say that two words u and u' from a sample S are *k-distinguishable* if there is a suffix z of length at most k such that $uz \in S_+$ and $u'z \in S_-$, or vice versa. Otherwise u and u' are *k-indistinguishable*.

Let (S_+, S_-) be an m-complete sample. Let d and k be natural numbers such that $m = d + k$. We say that two words u and u' from S are *(d, k)-equivalent*, denoted $u \equiv_{(d,k)} u'$, if u and u' are k-indistinguishable, $|u| \leq d$, and $|u'| \leq d$. We say that a sample (S_+, S_-) is *(d, k)-closed* if for every u from S with $|u| = d$ there is u' from S with $|u'| < d$ such that u and u' are k-indistinguishable.

Let $[u]_{\equiv_{(d,k)}}$ denote the equivalence class induced by $\equiv_{(d,k)}$ which contains the word u.

In Figure 2.3, the partitioning into equivalence classes induced by $\equiv_{(d,k)}$ is defined for $d = 3$ and $k = 1$. It turns out that there are 3 equivalence classes. Nodes are labeled by numbers which represent equivalence classes. The sample can be organised into an observation table. All prefixes of length at most d can be rows of the table, and all suffixes of length at most k can be columns of the table, but in contrast to L^* algorithm the number of entries of the observation table is exponential in d.

A DFA $\mathcal{H} = \langle \Sigma, L, l_0, \delta, L^f \rangle$ is constructed from m-complete and (d, k)-closed sample (S_+, S_-) where

- $L = \{[u]_{\equiv_{(d,k)}} | u \in S, |u| < d\}$,
- $l_0 = [\lambda]_{\equiv_{(d,k)}}$,

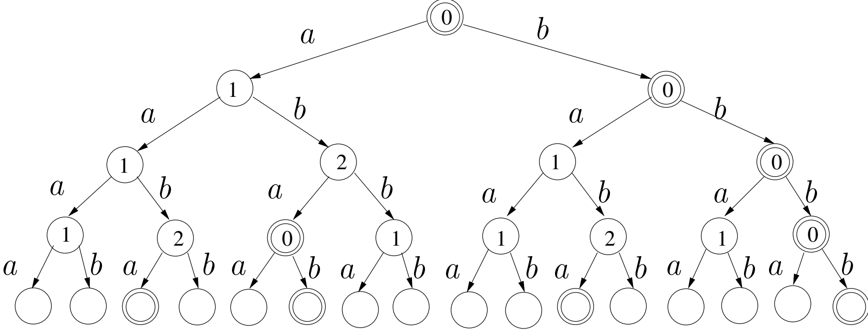


Figure 2.3: A tree for a 4-complete sample

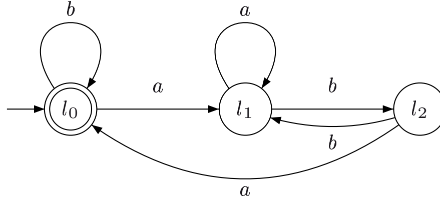


Figure 2.4: An automaton $\mathcal{H}$

- $\delta([u]_{\equiv_{(d,k)}}, a) = [ua]_{\equiv_{(d,k)}}$, where $a \in \Sigma$ and $|u| < d$, and
- $L^f = \{[u]_{\equiv_{(d,k)}} | u \in S_+, |u| < d\}$.

A state in $\mathcal{H}$ is an equivalence class induced by $\equiv_{(d,k)}$. In Figure 2.4 is shown $\mathcal{H}$ for a 4-complete sample shown in Figure 2.3.

Trakhtenbrot and Barzdin show that if $\mathcal{A} = \langle \Sigma, L, l_0, \delta, L^f \rangle$ has n states then the automaton $\mathcal{H}$ constructed from an $d + k$-complete sample, where

- $k \leq n - 2$ and any two words w and w' with $\delta(l_0, w) \neq \delta(l_0, w')$ are k-distinguishable,
- $d \leq n$ and any state is accessible from the initial state of $\mathcal{A}$ by a word of length at most $d - 1$,

is equivalent to $\mathcal{A}$. It follows that $\mathcal{A}$ can be constructed from $2n - 2$-complete sample. To obtain $2n - 2$ complete sample we need to ask $\frac{(|\Sigma|^{2n-1}-1)}{(|\Sigma|-1)}$ membership queries.

3. Timed systems

Timed systems differ from untimed systems in that their behavioral correctness relies not only on the results of their computations, but also on the actual times when the results are produced [Wan04]. Time can be modelled in different ways, e.g as discrete, continuous. We consider systems where time is modelled as continuous, that is the time domain is the nonnegative reals. One of the most popular model for timed systems is *timed automata* [AD94]. A timed automaton is an extension of a finite automaton by clocks that model timing. In this section we define informally timed automata and one of its subclasses, event-recording automata.

3.1 Timed and Event-Recording Automata

Let us consider the timed automata specification of a mouse double-click detector [NS03] in Figure 3.1. In the initial state the automaton occupies the location l_0, and clock x is zero. The value of clock x increases autonomously as time passes. When the user presses the mouse button (click), the automaton moves to the location l_1 and simultaneously resets clock x to zero. If the user clicks again before the clock x reaches the value 2, the automaton moves to location l_3, and then signals the detection of a double-click (doubleClick) after which it moves back to the initial location. If no click is made before 2 time units has elapsed after first click, the automaton moves back to the initial state.

Informally, a timed automaton is a finite graph equipped with a set of real-valued clocks C. The vertices of the graph are called locations, and edges are transitions from one location to another location. A clock can be reset to zero simultaneously with any transition. All clocks advance at the same rate and measure the amount of time that has elapsed since they were started or reset. Each edge is labeled with a *clock guard*, a symbol from Σ, and a set of clocks

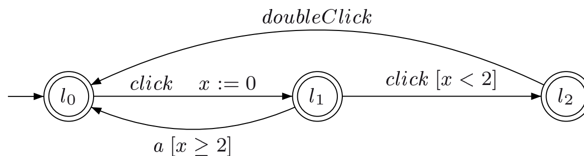


Figure 3.1: A timed automaton for a mouse double click detector

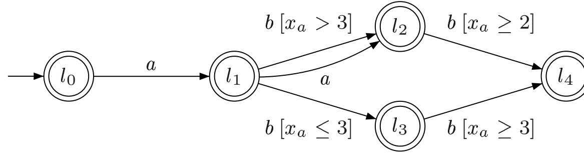


Figure 3.2: An event-recording automaton D_1

that are reset. A clock guard is a conjunction of atomic constraints of the form $x \sim k$ for $x \in C, \sim \in \{<, \leq, >, \geq\}$, where k is a natural number. It is required that the transition may be taken only if the current values of the clocks satisfy the clock guard. In Figure 3.1 the clock x is reset when the user presses the mouse button for the first time. The resetting of the clock x allows to model the behavior, where the double-click can occur if the second click occurs less than 2 time units after the first click.

A timed automaton D accepts a *timed language* $\mathcal{L}(D)$ that is a set of *timed words*. A *timed word* over Σ is a finite sequence $(a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ of symbols $a_i \in \Sigma$ paired with nonnegative real numbers t_i such that the sequence $t_1 t_2 \ldots t_n$ is nondecreasing. For example, the meaning of the timed word $(approach, 1)(lower, 2)$ is that the signal $lower$ occurs 1 time unit after the signal $approach$ occurs.

Important properties of timed automata which make it difficult to extend regular inference to timed automata is that we do not know how many clocks timed automata have and that clocks can be reset at any transition. It is not clear how to identify equivalent locations based on some generalization of Nerode's right congruence if guards in transitions from locations use different clocks. It is difficult to know which clocks to introduce. Therefore we move our attention to event-recording automata [AFH99], a subclass of timed automata. The difference between event-recording automata and timed automata is the use of clocks. An event-recording automaton contains for every symbol $a \in \Sigma$ a clock x_a, called the *event-recording clock* of a. A clock x_a records the time elapsed since the last occurrence of the symbol a. Then at each symbol, all clock values of the event-recording automata are determined by a timed word. Thus, if we know Σ, we know how many clocks an ERA has and when they are reset, in contrast to the case for timed automata. ERAs can be determinized, while timed automata in general cannot. For these reasons we use ERA as a representation of timed languages and study the problem of learning deterministic event-recording automata. A *deterministic event-recording automaton* (DERA) is an event-recording automaton such that if a location in the automaton has two different a successors, then they have nonoverlapping clock guards. DERA is more powerful than another popular model for real-time computation, timed transition system [AFH99].

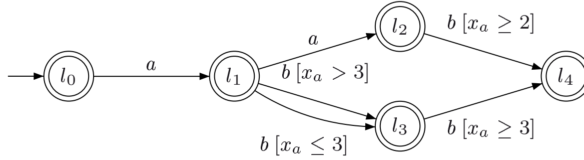


Figure 3.3: An event-recording automaton D_2

A timed automaton shown in Figure 3.1 can be understood as a DERA if we associate clock x with action $click$ and reset x also when second $click$ occurs. Another example of a DERA is shown in Figure 3.2. The DERA has two clocks x_a, x_b, since $\Sigma = \{a, b\}$, but guards use only clock x_a. Location l_0 is the start location of the automaton. All locations of the automaton are accepting. The clock constraint $x_a \geq 3$ that is associated with the edge from location l_3 to l_4 ensures that the action b can only be taken at least three time units after taking the transition from l_0 to l_1.

In the next section we show that deterministic event-recording automata do not have some properties that deterministic finite automata has.

3.2 Problems with DERA

Although DERAs have some nice properties that are not shared by timed automata in general, there are still problems with DERAs which influence the development of inference algorithms.

One problem is that it is not clear how to generalize Nerode's right congruence to the timed setting. Another problem is that DERAs do not have canonical form. Let us try to illustrate these problems in timed setting.

In the untimed case, Nerode's right congruence is used to let equivalent prefixes, i.e., prefixes that agree on the set of their suffixes, lead to the same state. For DERAs, it is natural to let a prefix correspond to a *guarded word*, i.e., a sequence of symbols paired with clock guards. For example in the DERA D_1 shown in Figure 3.2 the guarded word $(a, true)(b, x_a > 3)$ leads to the location l_2. In order to construct a DERA from a set of timed words, we need to combine timed words into guarded words and decide which guarded words should lead to the same location in DERA.

Consider the DERA D_1 shown in Figure 3.2. We can reach location l_2 in D_1 either by the guarded word $(a, true)(a, true)$, or by the guarded word $(a, true)(b, x_a > 3)$. However, these guarded words do not agree on the set of suffixes. If we reach location l_2 by the guarded word $(a, true)(a, true)$, then we can take the b transition from location l_2 when the value of clock x_a is 2, but if we reach location l_2 by the guarded word $(a, true)(b, x_a > 3)$ then we cannot take the b transition from location l_2 when the value of clock x_a is 2.

Thus, the two prefix guarded words $(a, true)(a, true)$ and $(a, true)(b, x_a > 3)$ are distinguished by the suffix $(b, x_a = 2)$ and according to Nerode's right congruence they should not be merged. However, they lead to the same location in the DERA D_1.

In general there is no canonical DERA that represents a given timed language. For example the DERA D_1 shown in Figure 3.2 and the DERA D_2 shown in Figure 3.3 (two b transitions from l_1 to l_2 can be merged into one b transition) are both smallest DERAs that accept the same timed language. It follows that a set of timed words can be represented by different sets of guarded words.

Our approach to overcoming the problem that DERAs have no canonical form is to focus on subclasses of DERAs that have canonical representation. By choosing subclasses that can be understood as DFAs, we can use Angluin's algorithm to learn them. However, these subclasses of DERAs can be exponentially bigger than the smallest DERAs accepting the same timed language. Therefore we also study the problem of learning DERAs without constructing a canonical representation. We must then develop a more sophisticated technique of inferring clock guards on transitions of DERAs and identifying equivalent guarded words. In the next chapter we discuss the algorithms presented in this thesis in more detail.

4. Contribution

4.1 Paper A

In Paper A we present three algorithms, TL^*_{sg}, TL^*_s, TL^*_{nsg} for learning of different classes of deterministic event-recording automaton (DERA). All algorithms use a table for organising queries in a similar manner as L^*, however the details are different.

The termination of Angluin's algorithm for learning DFA is guaranteed since DFAs have the canonical form. The problem with deterministic event-recording automata is that there is no unique minimal DERA. Therefore we work with subclasses of DERA which have a canonical form and can be understood as DFA over $\Sigma \times G$, where G is set of clock guards. The L^* algorithm can be used for learning corresponding DFAs and then interpret them as DERAs. In timed settings the $Learner$ asks membership queries in terms of guarded words. Since the $Teacher$ answers membership queries in terms of timed words, we extend the L^* algorithm with an *Assistant*. The role of the *Assistant* is to answer a membership query for a guarded word, posed by the $Learner$, by asking membership queries for timed words to the $Teacher$.

We present two subclasses of DERAs that have canonical forms: *event-deterministic ERAs* (EDERAs) and *simple DERAs*. In an EDERA each location has at most one outgoing transition per symbol from Σ. The canonical form of EDERAs is called sharply-guarded EDERA (SGEDERA). The TL^*_{sg} algorithm learns SGEDERA that can be exponentially bigger than smallest equivalent EDERA. Therefore we develop the algorithm TL^*_{nsg} that learns an EDERA without explicitly constructing a SGEDERA. The algorithm TL^*_{nsg} can learn a smaller automaton than TL^*_{nsg}, but can ask more membership and equivalence queries than TL^*_{sg}. TL^*_{nsg} introduces also an NP-hard problem to construct a smallest hypothesized EDERA. The TL^*_{sg} algorithm performs $O\left(kn^2ml\binom{|\Sigma|+K}{|\Sigma|}\right)$ membership queries and at most n equivalence queries, where $k = |\Sigma \times G|$, n is the number of locations in SGEDERA, m is the length of longest counterexample, l is the length of the longest guarded word queried, and K is the greatest constant which appears in guards of SGEDERA.

A simple DERA contains simple clock guards whose conjuncts are only of the form $x_a = n$ or $n < x_a \wedge x < n + 1$, $n \in \mathbb{N}$ and $x_a \in C$. Any DERA can be transformed into simple DERA which is at most exponentially bigger. Theoretically SGEDERA can be bigger than simple EDERA, but in practice they are usually smaller. A simple DERA can also be understood as a DFA

over $\Sigma \times G_s$, where G_s is a set of simple clock guards, which hence accepts simple guarded words (sequences of symbols that are paired with simple clock guard). We introduce the algorithm TL_s^* for learning simple DERAs using an *Assistant*. It is shown in the paper that the *Assistant* needs to ask only one membership query for a timed word to the $Teacher$ in order to be able to answer membership query from the $Learner$ for a simple guarded word. The TL_s^* algorithm performs $O(|\Sigma|^2 n^2 m K)$ membership queries and at most n equivalence queries, where n is the number of locations in simple DERA, m is the length of longest counterexample, and K is the greatest constant which appears in guards of simple DERA.

4.2 Paper B

The drawback of the algorithm TL_s^* presented in Paper A is that it constructs a DERA which has blow-up in transitions and is exponentially bigger than smallest equivalent DERA. This is due to the fact that the algorithm TL_s^* is based on a region graph, which is very sensitive to the largest constant in clock guards K. In Paper B we present an algorithm for learning DERAs, which avoids the construction of a region graph and need not have blow-up in transitions. Theoretically in the worst case the algorithm presented in Paper B is worse than TL_s^*, but we expect that in the average case it is better.

In Paper B we present an algorithm which constructs a tree from timed words for which queries have been performed and constructs a DERA by merging nodes of the tree. Then an equivalence query is performed and the tree is restructured based on the counterexample.

In Paper B we use a new data structure for organising results of queries, which we call a timed decision tree. Each edge of a timed decision tree is labeled by a symbol paired with a clock guard. Then every path in a timed decision tree is a guarded word and every query (timed word) follows some path. For each node in a timed decision tree we can calculate postcondition, a constraint on reachable clock values.

The problem is how to introduce clock guards into the tree. If there are two timed words that follow the same path in a timed decision tree, but one is accepted and other is rejected, then we should separate them. It can be the case that there are several clock guards which separate them. In the paper we present a procedure which searches for similar timed words, one accepted and one rejected, which can be separated only by one guard. For example, the timed words $(a, 0.6)(b, 1)$ and $(a, 0.6)(b, 1.1)$ can be separated only by the clock guard $x_b \leq 1$.

To construct a hypothesized DERA from a timed decision tree we need to merge nodes. We introduce a unification relation for merging nodes. The unification relation requires that a subtree of one node is "included" in the subtree of the other node. A problem with the unification relation is that it does not

guarantee that there is a bound on the size of a hypothesized DERA. The reason is that even if two nodes have different subtrees which cannot be merged by the unification relation, it may not be possible to find a separating suffix, since the postcondition of each node restricts the suffixes of the node. Hence it is possible that they correspond to the same location in the automaton to be learned. Therefore we introduce a second relation for merging nodes, which requires that postconditions of two nodes intersect the same regions (sets of equivalent clock valuations) and that subtrees of these nodes agree on suffixes that can actually occur in queries. A drawback of second relation, is that nodes, whose postconditions do not intersect the same regions, can not be merged. In contrast, the first unification relation often allows to merge nodes with postconditions which do not intersect the same regions. In order to combine advantages of the first and second relations, and generate automata that are always of bounded size and often small, we therefore introduce a "copying" operation, which restructures a tree with the goal that nodes that can be merged based on the second merging relation, can also be merged by the first.

The algorithm performs $O((\frac{|\Sigma|+2K+1)e}{|\Sigma|})^{|\Sigma|(m+1)})$ queries, where m is the length of the longest counterexample, e is the base of natural logarithm and K is the greatest constant which appears in guards of DERA.

5. Related Work

In this section we review results on learning of time-dependent systems.

Learning techniques were applied to different automata-like computation structures.

Verwer et. al [VdWW06] present an algorithm for passive learning of timed automata with one clock which is reset at every transition. Passive learning even for this class of timed automata is a hard problem, since one must decide how to organize timed words into guarded words. The algorithm constructs a prefix tree from a timed sample and then tries to merge nodes of this tree pairwise to form an automaton. If the resulting automaton does not agree with the sample then the last merge is undone and a new merge is attempted. The algorithm does not construct timed automata in a systematic way, and it is hard to generalize the algorithm to timed automata with more than one clock.

Huselis and Andersson [HA05] show how to synthesize a probabilistic state-machine model expressed in the ART-ML language from observations of implemented real-time system. A real-time system is a system of tasks that each executes jobs. For each job of a task, sequences of actions are obtained. Then a tree of actions is constructed which represents sequences. From the tree ART-ML model is constructed.

Some algorithms for learning untimed systems can also be used in a timed setting. The learning of finite state machines from given traces was introduced by Koskimies and Mäkinen [KM94]. States of finite state machine are associated with actions. An action is either a continuous activity ending when leaving the state, or a finite sequence of operations ending when the operations are completed. The set of actions is known. The algorithm starts with the assumption that the number of states of the state machine to be learned is equal to the number of actions. Then the algorithm tries to associate a state with every prefix. If there is no way to associate prefixes with states using the allowed number of states, the algorithm increases the number of states. Koskimies and Mäkinen apply the algorithm to learn a traffic-light controller. The timing behavior of the controller was modelled using time-out events which the timer raises when a certain time has elapsed since the previous light setting.

Sen et al. [SVA04] present an algorithm to learn Continuous Time Markov Chains from sample executions of a system. Continuous Markov Chains differ from DFAs in that a rate is associated with every transition in a Continuous Markov Chain. The algorithm constructs a prefix-tree continuous-time Markov chain from a sample and then merges equivalent states. An equivalence be-

tween states is defined on the basis of statistical tests. The algorithm checks for two states l and l' whether

- the total rate at which any transition from the state l is taken and the total rate at which any transition from the state l' is taken are equal within some statistical uncertainty, and
- the probabilities of taking the edge a from states l and l' are equal within some statistical uncertainty.

Learning techniques, such as reinforcement learning and classification, have been applied to time-dependent systems.

Reinforcement learning was applied to one of the problems in wireless communication, dynamic channel allocation [FJJ92]. Learning is used to minimize a cost function of channel use. The channel activity is time-dependent. In reinforcement learning, the learner is an agent that takes actions in an environment and receives reward or penalty for its action. The goal is to learn a sequence of actions that maximizes the total reward. Judy et al. [FJJ92] present approach where an agent learns to allocate two channels. The channel 2 costs more to use than channel 1. The agent is punished if channel 2 is busy and channel 1 is not, and is rewarded otherwise.

To study how data changes over time, time series have been introduced. A time series is a sequence of real-values, each one representing the value of a magnitude at a point of time. The series can be obtained by means of: recording the values of a magnitude in a real system or from a model simulation. Wei et al. [WK06] use learning from labeled and unlabeled data for time series classification. A classifier is one-nearest neighbor with Euclidean distance. The classifier is trained on the set of instances, where a labeled instance is a positive example. Then the classifier is used to classify unlabeled data as positive or negative.

A related field to this thesis is conformance testing for real-time systems. It was shown in [BGJ$^+$05] that if a set of examples form a conformance test suite for a finite state machine M then M can be inferred from these set of examples using automata learning techniques. It was also shown that if finite state machine M is inferred from the set of examples, and furthermore M is the only such automaton, then the set of examples forms a conformance test suite for M. Springintveld et al. [SVD01] introduces an algorithm which generates a conformance test suite for timed automata. The algorithm constructs grid automata, which only contain states in which every clock value is from the set of integer multiples of 2^{-n} for some sufficiently large natural number n. Then the Vasilevskii-Chow algorithm [Cho78], [Vas73] is applied to the grid automaton to generate a conformance test suite.

6. Conclusion and Future Work

6.1 Conclusion

Developing algorithms for learning DERAs requires dealing with three problems. One problem is that there is no Nerode's right congruence for DERA. Another problem is that there is no canonical form for DERAs. The third problem is that the procedure is needed for identification of clock guards in DERA. This thesis presents subclasses of DERA, EDERA and simple DERA, which have the canonical form. The canonical form of EDERA has a form of the zone graph and the canonical form of simple DERA has a form of the region graph. Every DERA can be transformed to simple DERA. However, since zone graph is usually smaller than region graph, learning of EDERA is interesting problem. This thesis also introduces two different ways of learning clock guards in DERAs.

In the thesis four algorithms are presented for DERAs. Two algorithms learn a subclass of event-recording automata, EDERAs, and two algorithms learn DERAs. It is shown that the L^* algorithm can be applied both for learning EDERAs and DERAs. A problem is that in the timed setting the L^* algorithm learns a language, whose elements are guarded words. Thus, the $Learner$ asks membership queries in terms of guarded words. Since the elements of a timed language are timed words, the $Teacher$ answers membership queries in terms of timed words. Therefore we introduce an *Assistant* who learns guarded words by asking membership queries for timed words to the $Teacher$.

A drawback of using the L^* algorithm for EDERA is that it can produce EDERAs which are exponentially bigger than the automaton to be learned. In the thesis, we present a modification of the L^* algorithm for learning EDERA, which constructs the smallest EDERA equivalent to the automata to be learned. However, this algorithm requires to solve NP-hard problem in order to find rows in an observation table which represent the same location in the EDERA.

A drawback of using the L^* algorithm for DERA is that it produces DERAs in the form of region graph. Therefore we present an algorithm for learning DERAs which uses a new data structure, a timed decision tree. We develop a procedure for introducing clock guards into the tree and for merging nodes of the tree. Since a clock guard is not always unique, and nodes can be merged in different ways, the algorithm can produce exponentially bigger DERA than the automaton to be learned. In the worse case, this algorithm can produce a DERA which is bigger than a DERA constructed by the L^* algorithm. How-

ever, we believe that in the average case the algorithm works better than the L^* algorithm.

One limitation of the algorithms in this thesis is that they work under assumption that a greatest constant K which appears in clock guards is known. This assumption guarantees termination of the algorithms. To find an algorithm which does not require this assumption is a difficult problem. One way can be in choosing some value for K, and then wait for a counterexample which shows that K must be increased. However, we have not been able to find a guarantee that we will not construct bigger and bigger DERA for the same K.

6.2 Future work

Theoretically, the algorithm for learning DERAs using timed decision trees has a higher worst case complexity than the approach based on the L^* algorithm, but we expect better performance in the average case. Thus, it would be interesting to implement these algorithms and compare their performance on some examples. Another problem is that the algorithms assume that the greatest constant K is known. To develop algorithms which do not require this assumption is also an interesting problem.

The algorithms in this thesis have high complexity. However, we can modify the algorithm based on timed decision trees in such a way that it can learn DERAs with one clock by asking only a polynomial number of membership and equivalence queries. This is possible, since every node in a timed decision tree with one clock has the same postcondition. Thus, we can compare all nodes using the same set of suffixes and the number of locations in a DERA constructed from a timed decision tree is at most n, where n is the number of locations in a smallest DERA equivalent to automaton to be learned. If a split need to be introduced into a timed decision tree with one clock, an unique clock guard can be found by asking polynomial number of membership queries.

It would be interesting to find a subclass of DERA with two clocks for which a smallest automaton can be inferred by asking polynomial number of queries. The problem is that in general there is no unique split to be introduced into the timed decision tree to make the tree consistent. It can be shown that for DERAs which contain only clock guards whose conjuncts of the form $x_a < n$ or $x_a \geq n$ splits are unique. However it can be the case that introducing different splits into the timed decision tree, not found by the splitting procedure, can create a smaller DERA. It follows that finding subclass of DERA with two clocks which can be inferred with better complexity is not easy problem. A difference between timed automata with one and two clocks is shown in [LMS04]. Laroussinie et al. present the polynomial algorithm for model checking $TCTL_{\leq,\geq}$ over timed automata with one clock and show that model checking CTL over timed automata with two clocks is PSPACE-complete.

DERA is a subclass of deterministic timed automata. How to perform learning of deterministic timed automata is also an interesting and hard problem. In the case of DERAs, if we know the alphabet we know how many clocks the DERA has and when they are reset. In contrast, we do not know how many clocks a timed automaton has and they can reset at any transition. We only know that it does not make sense to have two clocks that reset at the same transitions. We can construct a timed decision tree by introducing at every transition a new clock, but then it is not clear how a merging procedure should work for constructing a timed automaton from the timed decision tree, since we need to compare edges which are labeled by guards containing different clocks.

In the thesis we assume that the model which we learn is fully observable. Systems are often built by composition of many components. Then the interaction between components is not observable externally. In our learning algorithms we assume that the time of inputs can be observed precisely. In real systems, clocks can be drift and measure time only up to some precision. Thus, the problem of learning partially observable timed system and timed system with drift in clocks should be studied.

Bibliography

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH99] R. Alur, L. Fix, and T. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[Alp04] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

[AMN05] Rajeev Alur, P. Madhusudan, and Wonhong Nam. Symbolic compositional verification by learning assumptions. In *Proc. of International Conference on Computer Aided Verification*, pages 548–562, 2005.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[BDG97] Jos L. Balczar, Josep Daz, and Ricard Gavald. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity*, pages 53–72, 1997.

[BGJ+05] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In *Proc. of 8th International Conference on Fundamental Approaches to Software Engineering*, pages 175–189, 2005.

[BO05] Miguel Bugalho and Arlindo Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, 38(9):1457–1467, 2005.

[CGP03] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In *Proc. of International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346, 2003.

[Cho78] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978.

[FJJ92] Smith Michael Franklin Judy and Yun Jay. Learning channel allocation strategies in real time. In *Proc. IEEE Vehicular Technolgy Conference*, pages 768–772, 1992.

[Gol78] Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[HA05] Joel Huselius and Johan Andersson. Model synthesis for real-time systems. In *Proceedings of 9th European Conference on Software Maintenance and Reengineering*, pages 52–60, 2005.

[ISBF07] Kenneth L. Ingham, Anil Somayaji, John Burge, and Stephanie Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.

[KM94] Kai Koskimies and Erkki Makinen. Automatic synthesis of state machines from trace diagrams. *Software-Paractice and Experience*, 24(7):643–658, 1994.

[Kun01] Peep Kungas. Learning state machines in the robot moving context. In J. Grundspenkis K. Wang and A. Yerofeyev, editors, *Lecture Notes of the Nordic, Baltic and Northwest Russian Summer School on Applied Computational Intelligence to Engineering and Business*, 2001.

[Leu07] Martin Leucker. Learning meets verification. In *Proc. of International Symposium on Formal Methods for Components and Objects*, pages 1–21, 2007.

[LGF00] Steve Lawrence, C. Lee Giles, and Sandiway Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Trans. Knowl. Data Eng.*, 12(1):126–140, 2000.

[LGS06] Keqin Li, Roland Groz, and Muzammil Shahbaz. Integration testing of components guided by incremental state machine learning. In *Testing: Academia and Industry Conference - Practice and Research Techniques*, pages 59–70, 2006.

[LMS04] Franois Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of 15th International Conference on Concurrency Theory*, pages 387–401, 2004.

[Mur96] K. Murphy. Passively learning finite automata. Technical report, Santa Fe Institute, 1996.

[NS03] Brian Nielsen and Arne Skou. Automated test generation from timed automata. *Software Tools for Technology Transfer*, 5(1):59–77, 2003.

[PH01] Rajesh Parekh and Vasant Honavar. Learning dfa from simple examples. *Machine Learning*, 44(1/2):9–35, 2001.

[PNH98] Rajesh Parekh, Codrin M. Nichitiu, and Vasant Honavar. A polynominal time incremental algorithm for learning dfa. In *Proc. of International Colloquium on Grammatical Inference: Algorithms and Applications*, pages 37–49, 1998.

[PVY99] Doron Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Formal Methods for Protocol Engineering and Distributed Systems, FORTE/PSTV*, pages 225–240, Beijing, China, 1999. Kluwer.

[RS93] R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning continious time markov chains from sample executions. In *Proc. International Conference on Qualitative Evaluation of Systems*, pages 146–155, 2004.

[SVD01] Jan Springintveld, Frits W. Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theor. Comput. Sci.*, 254(1-2):225–257, 2001.

[TB73] B.A. Trakhtenbrot and J.M. Barzdin. *Finite automata: behaviour and synthesis*. North-Holland, 1973.

[Utg86] P. E. Utgoff. Shift of bias for inductive concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 107–148. Kaufmann, Los Altos, CA, 1986.

[Vas73] M. P. Vasilevskii. Failure diagnosis of automata. *Cybernetic*, 9(4):653–665, 1973.

[VdWW06] Sicco E. Verwer, Mathijs M. de Weerdt, and Cees Witteveen. Identifying an automaton model for timed data. In *Proc. of Machine Learning Conference of Belgium and the Netherlands*, pages 57–64, 2006.

[Wan04] Farn Wang. Formal verification of timed systems: a survey and perspective. *Proceedings of IEEE*, 92(8):1283–1307, 2004.

[WK06] Li Wei and Eamonn J. Keogh. Semi-supervised time series classification. In *Proc. of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 748–753, 2006.

[YIK94] Takashi Yokomori, Nobuyuki Ishida, and Satoshi Kobayashi. Learning local languages and its application to protein α-chain identification. In *Proc. of Hawaii International Conference on System Sciences*, pages 113–122, 1994.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 434

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2008

Distribution: publications.uu.se
urn:nbn:se:uu:diva-8763