Alexander Clark
François Coste
Laurent Miclet (Eds.)

LNAI 5278

# Grammatical Inference: Algorithms and Applications

9th International Colloquium, ICGI 2008
Saint-Malo, France, September 2008
Proceedings

Springer

Alexander Clark François Coste
Laurent Miclet (Eds.)

# Grammatical Inference: Algorithms and Applications

9th International Colloquium, ICGI 2008
Saint-Malo, France, September 22-24, 2008
Proceedings

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Alexander Clark
Royal Holloway, University of London
Department of Computer Science
Egham, Surrey TW20 0EX, UK
E-mail: alexc@cs.rhul.ac.uk

François Coste
INRIA Rennes - Bretagne Atlantique
IRISA/Symbiose
Campus de Beaulieu, 35042 Rennes CEDEX, France
E-mail: Francois.Coste@irisa.fr

Laurent Miclet
University of Rennes 1
IRISA/ENSSAT
22305 Lannion CEDEX, France
E-mail: miclet@enssat.fr

Library of Congress Control Number: 2008935398

CR Subject Classification (1998): I.2, F.4, F.3

LNCS Sublibrary: SL 7 – Artificial Intelligence

| | |
|---|---|
| ISSN | 0302-9743 |
| ISBN-10 | 3-540-88008-9 Springer Berlin Heidelberg New York |
| ISBN-13 | 978-3-540-88008-0 Springer Berlin Heidelberg New York |

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12525829 06/3180 5 4 3 2 1 0

# Preface

The 9th International Colloquium on Grammatical Inference (ICGI 2008) was held at the Palais du Grand Large, Saint Malo, France during September 22–24, 2006. ICGI 2008 was the ninth in a series of successful biennial international conferences in the area of grammatical inference. Previous meetings were held in Essex, UK; Alicante, Spain; Montpellier, France; Ames, Iowa, USA; Lisbon, Portugal; Amsterdam, The Netherlands; Athens, Greece; Tokyo, Japan. This series of conferences seeks to provide a forum for presentation and discussion of original research papers on all aspects of grammatical inference.

Grammatical inference, the study of learning grammars from data, is an established research field in artificial intelligence, dating back to the 1960s and has been extensively addressed by researchers in automata theory, language acquisition, computational linguistics, machine learning, pattern recognition, computational learning theory and neural networks. ICGI 2008 particularly emphasized the multi-disciplinary nature of the research field and the diverse domains in which grammatical inference is being applied, such as natural language acquisition, computational biology, structural pattern recognition, information retrieval, Web mining, text processing, data compression and adaptive intelligent agents.

We received 36 high-quality papers from 15 countries around the world. The papers were reviewed by three reviewers. Based on the positive comments of the reviewers, 21 full papers were accepted. In addition, we decided to accept eight short papers for poster presentation. Short papers appear as extended abstracts in a separate section of this volume. The topics of the accepted papers vary from theoretical results of learning algorithms to innovative applications of grammatical inference, and from learning several interesting classes of formal grammars to applications to natural language processing.

The editors would like to acknowledge the contribution of the conference's Program Committee and the additional reviewers in reviewing the submitted papers, and we thank the Organizing Committee for their invaluable help in organizing the conference. We would also like to acknowledge the use of the EasyChair conference system, in the submission and reviewing process. Finally, we are grateful for the generous support and sponsorship of the conference by the PASCAL2 Network of Excellence, the INRIA, the University of Rennes 1 and the Région Bretagne.

July 2008

Alexander Clark
François Coste
Laurent Miclet

# Organization

## Program Chairs

Alexander Clark
François Coste
Laurent Miclet

## Program Committee

Pieter Adriaans
Dana Angluin
Jose Balcazar
Rens Bod
Rafael Carrasco
Christophe Costa Florencio
Francois Denis
Pierre Dupont
Remi Eyraud
Henning Fernau
Remi Gilleron
Vasant Honavar
Makoto Kanazawa
Satoshi Kobayashi
Leonid Kontorovich
Tim Oates
Jose Oncina
Georgios Paliouras
Rajesh Parekh
Yasubumi Sakakibara
Isabelle Tellier
Franck Thollard
Etsuji Tomita
Enrique Vidal
Chris Watkins
Ryo Yoshinaka
Colin de la Higuera
Menno van Zaanen

## Local Organization

Elisabeth Lebret
Marie-Noëlle Georgeault
Matthias Gallé

## External Reviewers

Tom Armstrong
Rafael Carrasco
Amaury Habrard
Gabriel Infante-Lopez
Jean-Christophe Janodet
Satoshi Kobayashi
Stasinos Konstantopoulos
Faissal Ouardi
Georgios Paliouras
Georgios Petasis
Emilie Samuel
Kengo Sato
Yasuhiro Tajima
Frederic Tantini
Franck Thollard
Enrique Vidal
Mitsuo Wakatsuki

## Sponsoring Organizations

# Table of Contents

# Learning Meaning Before Syntax

Dana Angluin and Leonor Becerra-Bonache*

Department of Computer Science, Yale University
P.O. Box 208285, New Haven, CT, USA
{dana.angluin,leonor.becerra-bonache}@yale.edu

**Abstract.** We present a simple computational model that includes semantics for language learning, as motivated by readings in the literature of children's language acquisition and by a desire to incorporate a robust notion of semantics in the field of Grammatical Inference. We argue that not only is it more natural to take into account semantics, but also that semantic information can make learning easier, and can give us a better understanding of the relation between positive data and corrections. We propose a model of meaning and denotation using finite-state transducers, motivated by an example domain of geometric shapes and their properties and relations. We give an algorithm to learn a meaning function and prove that it finitely converges to a correct result under a specific set of assumptions about the transducer and examples.

**Keywords:** semantics, finite-state transducers, corrections.

## 1 Introduction

In 1972, Feldman stated that some of the interesting remaining questions in Grammatical Inference were: *inference in the presence of noise, general strategies for interactive presentation and the inference of systems with semantics* [1]. Thirty-six years later some of these questions still remain open. In this paper, we focus on the last one: learning systems with semantics.

The results of Gold [2] show that not even regular grammars can be learned in the limit from positive data. Despite this, positive learnability results have been obtained by restricting the class of grammars to be learned, restricting the method of selecting examples, providing structural information, or making negative data available. See [3,4] for surveys of these results.

These approaches tend to omit semantic information and reduce the language learning problem to syntax learning. In natural situations, semantic information is also available to the child [5,6,7,8]. Moreover, semantic information plays an important role in the early stages of children's language acquisition, particularly in the two-word stage, when children go from the production of one word to the combination of two elements. Two-word sentences may be viewed as "semantic speech" [9,10]; the meanings of the two elements in the shared context

* Supported by a Marie Curie International Fellowship within the $6^{th}$ European Community Framework Programme.

indicate the implied syntactic relations. Despite their very different grammars, adult and child can communicate with each other because of their shared context and semantics. "Syntactic speech" develops when the communication is less contextually determined and the child uses more complex syntactic rules. Given the established difficulty of learning formal grammars, we conjecture that semantics can make language learning easier.

Several researchers have studied the role of semantics in language learning; the approaches nearest to ours are [11,12,13]. These approaches require a correct or nearly correct parse to assign a meaning to a sentence. In the case of two-word sentences, the utterances of the child are far from parsing in the adult grammar and vice versa. One goal of our model is to overcome this obstacle to communication. Moreover, in contrast to approaches that assume the learner's input is pairs consisting of an utterance and its meaning, the inputs of our learning algorithm are pairs consisting of an utterance and the situation in which the utterance is produced, analogous to the shared context of adult and child in the two-word stage.

Ideally, the input provided to the learner should be the same kinds of examples that are available to the child. Whereas the availability of positive data (i.e., utterances that are grammatically correct) is generally accepted, the availability of another kind of data, which is often called negative data, remains a matter of significant controversy. However, there is a kind of information that is sometimes available during the two-word stage, namely, a type of correction called an *expansion*. The following example is from Brown and Bellugi [14]):

CHILD: Eve lunch
ADULT: Eve is having lunch

The adult's answer is a grammatical sentence (positive information), but is an expansion of an incomplete sentence uttered by the child (providing negative information by suggesting that the child's utterance was not grammatically correct). The expansion *preserves the meaning* of the child's utterance; the adult has understood the child's meaning and re-expressed it in the adult grammar. The shared context is crucial – in other contexts the adult would reply differently. One goal of our model is to be able to reflect this kind of interaction.

Also, there is a close relationship between positive data and expansions. Consider the two cases depicted in Fig. 1. In case A, the child receives positive information and, using the context, determines the intended meaning, which the child then re-expresses in her grammar. (This kind of "echo" is sometimes explicitly verbalized [7,15].) In case B, the child produces a sentence in her grammar which is understood and expanded by the adult. In both cases the child has access to the meaning and to the adult and child expressions of it.

We propose a new computational learning model to reflect the issues of context, semantics, positive data and corrections. The model should accommodate at least two different tasks: comprehension and production. Here we focus on comprehension. The scenario we consider is cross-situational and supervised; the teacher provides to the learner several example pairs consisting of a situation and a utterance that denotes something in the situation. The goal of the

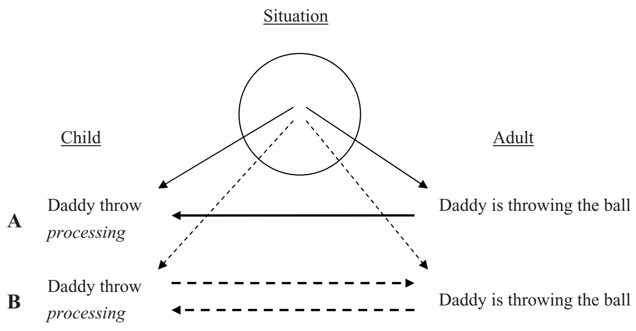


**Fig. 1.** Adult and child share a situation. In A, the child receives positive data. In B, an expansion is returned to the child.

learner is to learn the meaning function, permitting the comprehension of novel utterances. In Sect. 2, we define meaning and denotation functions; in Sect. 3, we examine simple learning strategies to motivate our algorithm to learn a meaning function, which we present and prove correct in Sect. 4. We conclude with a discussion in Sect. 5. More details are available in [16].

## 2 A Model of Meaning and Denotation

*Meaning Functions.* To specify a meaning function, we use a finite state transducer $M$ that maps sequences of words to sequences of predicate symbols, and a path-mapping function $\pi$ that maps sequences of predicate symbols to sequences of logical atoms. We consider three disjoint finite alphabets of symbols: $W$, the set of **words**, $P$, the set of **unary predicate symbols**, and $R$, the set of **primary binary predicate symbols**. For each symbol $r \in R$, there is also a new binary predicate symbol $r^t$, which is used to denote $r$ with its arguments reversed; the set of all such $r^t$ is denoted $R^t$. The symbols in $P$ and $R$ are **primary predicates**, and the symbols in $R^t$ are **derived predicates**. The function *primary* maps each primary predicate symbol to itself, and each predicate symbol $r^t$ to $r$.

An **utterance** is a finite sequence of words, that is, an element of $W^*$. Define the function $c$ to map a finite sequence of elements to the set of distinct elements occuring in the sequence. Thus, $c(u)$ is the set of words occurring in the utterance $u$.

We define a **meaning transducer** $M$ with input symbols $W$ and output symbols $Y = P \cup R \cup R^t$. $M$ has a finite set $Q$ of states, an initial state $q_0 \in Q$, a finite set $F \subseteq Q$ of final states, a deterministic transition function $\delta$ mapping $Q \times W$ to $Q$, and an output function $\gamma$ mapping $Q \times W$ to $Y \cup \{\varepsilon\}$, where $\varepsilon$ denotes the empty sequence.

The transition function $\delta$ is extended to define $\delta(q, u)$ to be the state reached from $q$ following the transitions specified by the utterance $u$. The **language** of $M$, denoted $L(M)$ is the set of all utterances $u \in W^*$ such that $\delta(q_0, u) \in F$. For each utterance $u$, we define the **output** of $M$, denoted $M(u)$, to be the finite sequence of non-empty outputs produced by starting at state $q_0$ and following the transitions specified by $u$. A state $q \in Q$ is **live** if there exists an utterance $u$ such that $\delta(q, u) \in F$, and **dead** otherwise.

As an illustration, we describe an extended example of utterances in English involving geometric shapes and their properties and relative locations. $W$ contains the words *the*, *triangle*, *square*, *circle*, *red*, *blue*, *green*, *above*, *below*, *to*, *left*, *right* and *of*. $P$ contains the symbols *tr*, *sq*, *ci*, *bi*, *re*, *bl*, *gr* referring to the properties of being a triangle, a square, a circle, big, red, blue, and green, respectively, and $R$ contains the symbols *ab*, *le*, referring to the relations of being above and to the left of, respectively. (Note that there is no word *big* – a property or relation may not have a corresponding word.) We define the meaning transducer $M_1$ as follows. $M_1$ has states $q_i$ for $0 \leq i \leq 7$; $q_0$ is the initial state and there is one final state, $q_2$. The transition function is partially defined in Fig. 2. Undefined transitions go to a non-final dead state, $q_7$. $L(M_1)$ contains such utterances as *the triangle*, *the blue triangle to the left of the red circle*. The output of $M_1$ for the utterance *the triangle* is the sequence $\langle tr \rangle$; the output of $M_1$ for the utterance *the blue triangle above the square* is the sequence $\langle bl, tr, ab, sq \rangle$.

*Path-mapping.* Given a finite sequence of predicate symbols, we define a specific function, path-mapping, to convert it into a finite sequence of atoms in the predicate logic. Let $x_1, x_2, \ldots$ be distinct variables and $t_1, t_2, \ldots$ be distinct constants. Different constants will be used to denote different objects in a situation. An **atom** is one of $p(v)$, where $p \in P$ or $r(v, w)$ or $r^t(v, w)$, where $r \in R$ and $v$ and $w$ are constants or variables. An atom is **primary** if its predicate symbol is primary, that is, not from $R^t$. An atom is **ground** if it does not contain any variables.

The **path-mapping function**, denoted $\pi$, takes a finite sequence of predicate symbols and supplies each predicate with the correct number of argument variables, as follows. Working left to right, $x_1$ is the argument of each predicate in the initial sequence of unary predicates, then $x_1$ and $x_2$ (in order) are the

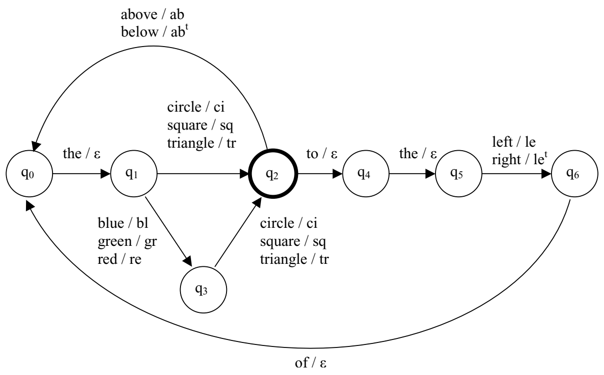


**Fig. 2.** Meaning transducer $M_1$

arguments of the first binary predicate, then $x_2$ is the argument of each of the subsequent sequence of unary predicates, then $x_2$ and $x_3$ (in order) are the arguments of the second binary predicate, and so on, introducing successive variables for successive binary predicates. Applying $\pi$ to the sequence of predicates

$$\langle \mathit{bl}, \mathit{tr}, \mathit{ab}, \mathit{sq}, \mathit{le}^t, \mathit{re}, \mathit{ci} \rangle,$$

we get the following sequence of atoms

$$\langle \mathit{bl}(x_1), \mathit{tr}(x_1), \mathit{ab}(x_1, x_2), \mathit{sq}(x_2), \mathit{le}^t(x_2, x_3), \mathit{re}(x_3), \mathit{ci}(x_3) \rangle.$$

The **meaning** assigned by a meaning transducer $M$ to an utterance $u$ is $\pi(M(u))$. As an example, the meaning assigned by $M_1$ to the utterance *the blue square to the right of the green circle* is

$$\langle \mathit{bl}(x_1), \mathit{sq}(x_1), \mathit{le}^t(x_1, x_2), \mathit{gr}(x_2), \mathit{ci}(x_2) \rangle.$$

The definition of $\pi$ reflects a strong restriction on the way properties and relations can be expressed by a meaning transducer.

*Situations and Denotation Functions.* Next we define situations, which represent the objects, properties and binary relations that are noticed in some environment of the teacher or learner. A **situation** is a finite set of primary ground atoms. [1]

Noticing a big blue triangle above a big green square gives the following situation.

$$S_1 = \{\mathit{bl}(t_1), \mathit{bi}(t_1), \mathit{tr}(t_1), \mathit{ab}(t_1, t_2), \mathit{gr}(t_2), \mathit{bi}(t_2), \mathit{sq}(t_2)\}.$$

The **things** in a situation $S$, denoted $\mathit{things}(S)$, is the set of all $t_i$ that occur in atoms in $S$. The assignment of constants $t_i$ to things in the situation is arbitrary. The **predicates** in a situation $S$, denoted $\mathit{predicates}(S)$, is the set of all predicate symbols that occur in atoms in $S$. Thus, $\mathit{things}(S_1) = \{t_1, t_2\}$ and $\mathit{predicates}(S_1) = \{\mathit{bl}, \mathit{bi}, \mathit{tr}, \mathit{ab}, \mathit{gr}, \mathit{sq}\}$.

To determine the denotation of an utterance $u$ in a situation $S$, the teacher takes the meaning $\pi(M(u))$ of $u$ and attempts to match it to a subset of the situation. A ground atom $A$ is **supported by** a situation $S$ if $A$ is primary and an element of $S$, or, if $A$ is $r^t(t_i, t_j)$ for some $r \in R$ and $r(t_j, t_i)$ is an element of $S$. For example, $\mathit{gr}(t_2)$, $\mathit{ab}(t_1, t_2)$, and $\mathit{ab}^t(t_2, t_1)$ are supported by the situation $S_1$ defined above, but $\mathit{gr}(t_1)$ and $\mathit{le}(t_1, t_2)$ are not.

Let $V = \{x_1, \ldots, x_k\}$ denote the variables that occur in $\pi(M(u))$ and $T$ denote the things in the situation $S$. A **match** of $\pi(M(u))$ to $S$ is a one-to-one function $f$ from $V$ to $T$ such that substituting $f(x_i)$ for every occurrence of $x_i$ in $\pi(M(u))$ produces a set of ground atoms that are all supported by the situation $S$. A match is **unique** if no other one-to-one function of $V$ to $T$ is also a match of $\pi(M(u))$ to $S$. Given a match $f$, the **first thing mentioned** is the constant $f(x_1)$ and the **last thing mentioned** is the constant $f(x_k)$.

[1] Only primary predicates (from $P \cup R$) occur in situations, although meanings may use both primary predicates and derived predicates (from $R^t$).

As an example, the function $f(x_1) = t_1$ and $f(x_2) = t_2$ is a unique match of $\pi(\langle bl, tr, ab, sq \rangle)$ in the situation $S_1$. In this match, the first thing mentioned is $t_1$ and the last thing mentioned is $t_2$.

A **denotation function** is specified by a meaning transducer $M$ and a choice of a parameter *which* from $\{\mathit{first}, \mathit{last}\}$. Given an utterance $u$ and a situation $S$ such that $u \in L(M)$ and there is a unique match $f$ of $\pi(M(u))$ in $S$, then the denoted object is the first thing mentioned if $\mathit{which} = \mathit{first}$ and the last thing mentioned if $\mathit{which} = \mathit{last}$. Otherwise, the denotation function is undefined for $u$ and $S$.

With $M_1$ we specify a denotation function by choosing $\mathit{which} = \mathit{first}$. Then in the situation $S_1$, the utterance *the blue triangle above the square* denotes $t_1$ and *the square below the blue triangle* denotes $t_2$. The utterance *the green triangle* has no denotation in the situation $S_1$.

## 3 Strategies for Learning Meanings

For the meaning function, we assume that the learner receives a sequence of pairs $(S_i, u_i)$ from the teacher, where $S_i$ is a situation, and $u_i$ is an utterance with a denotation in the situation $S_i$. For the denotation function, we assume that the learner receives triples $(S_i, u_i, d_i)$ where $S_i$ is a situation, $u_i$ is a denoting utterance in $S_i$, and $d_i$ indicates which thing, $t_j$, in the situation is denoted by $u_i$. In practice, the denoted object might be indicated by non-linguistic means, e.g., pointing at it. This setting gives rather less information than pairs consisting of an utterance $u \in L(M)$ and its meaning $\pi(M(u))$. We focus on learning the meaning function because once it is learned, learning the denotation function just requires setting the parameter *which* correctly. We assume that the learner and teacher share the relevant situation $S_i$.

Given a pair $(S, u)$ of situation and utterance, the learner knows that the teacher's transducer $M$ may map some words in $u$ to the empty result, $\varepsilon$, but each other word in $u$ must have been mapped by $\gamma$ either to one of the predicates in $S$, or to $r^t$, where $r$ is a binary predicate in $S$. In general, the mapping of a word by $M$ depends on the state that $M$ is in when the word is encountered. However, in the transducer $M_1$, the output map is state-independent, at least for states other than the dead state. To simplify the learning problem, in this paper we make the following **state-independence assumption**.

**Assumption 1.** *For all states $q, q' \in Q$ and words $w \in W$, $\gamma(q, w) = \gamma(q', w)$.*

Thus we write $\gamma(w)$ instead of $\gamma(q, w)$. Under this assumption, knowing $\gamma$ is sufficient to compute $M(u)$ for any $u \in L(M)$; we apply $\gamma$ sequentially to the words of $u$ and form the sequence of results. And because the path-mapping function $\pi$ is fixed, knowing $\gamma$ is sufficient to compute the meaning $\pi(M(u))$ of any $u \in L(M)$. Thus, we may think of $M$ as separated into a finite state acceptor for $L(M)$ and the function $\gamma$ to compute the outputs for elements of $L(M)$.

*A Cross-Situational Conjunctive Strategy.* Given the state-independence assumption, we consider a cross-situational conjunctive learning strategy. Cross-situational learning has been investigated by [17,18,19], among others. For each encountered word $w$, we consider all utterances $u_i$ containing $w$ and their corresponding situations $S_i$, and form the intersection of the sets of predicates occurring in these $S_i$. That is, for each encountered word $w$ let

$$C(w) = \bigcap \{predicates(S_i) : w \in c(u_i)\}.$$

Because $u_i$ is a correct denotation in $S_i$, if $\gamma(w)$ is a primary predicate, that predicate must be in $C(w)$. Similarly, if $\gamma(w) = r^t$, then $r$ must be in $C(w)$. Hence, if $C(w)$ is empty then the learner may correctly conclude that $\gamma(w) = \varepsilon$.

Continuing with our earlier example, suppose we apply this approach to the pairs of utterances and situations shown in the left part of the table below. Each situation is described by an abbreviation: for example *brtlbbs* represents the situation of a big red triangle to the left of a big blue square.

| utterance | situation |
|---|---|
| *the triangle* | *bbt* |
| *the blue triangle* | *bbtlbrt* |
| *the red triangle to the left of the blue square* | *brtlbbs* |
| *the circle above the green triangle* | *bbcabgt* |
| *the red circle to the right of the green circle* | *bgclbrc* |
| *the triangle above the red square* | *bgtabrs* |
| *the green triangle* | *bgtabrs* |
| *the blue circle* | *bbcabgt* |
| *the red triangle to the right of the blue triangle* | *bbtlbrt* |
| *the red circle* | *brc* |
| *the circle above the square* | *bbcabrs* |
| *the circle to the left of the square* | *bgclbgs* |
| *the blue circle above the square* | *bbcabgs* |
| *the circle to the left of the triangle* | *bbclbgt* |
| *the triangle to the right of the circle* | *bbclbgt* |

| $w$ | $C'(w)$ |
|---|---|
| *the* | $\emptyset$ |
| *triangle* | $\{tr\}$ |
| *circle* | $\{ci\}$ |
| *square* | $\{sq\}$ |
| *blue* | $\{bl\}$ |
| *red* | $\{re\}$ |
| *green* | $\{gr\}$ |
| *above* | $\{ab\}$ |
| *left* | $\{le\}$ |
| *right* | $\{le\}$ |
| *to* | $\{le\}$ |
| *of* | $\{le\}$ |

For the data in the left part of this table, every $C(w)$ contains the predicate *bi* because it is present in every situation; it is a **background predicate**. Let $C'(w)$ be $C(w)$ with all background predicates removed. The values of $C'(w)$ for this data are shown in the right part of this table. [2]

The results in this table for *the*, *triangle*, *circle*, *square*, *blue*, *red*, *green*, *above* and *left* agree with the meaning function for $M_1$, but the values for *right*, *to* and *of* disagree. For *right*, the value should be $le^t$ instead of $le$. This arises because although derived predicates may be meanings, they do not occur in situations. As an example of how we might determine that $ab$ rather than $ab^t$ is the correct image of the word *above*, consider the pair consisting of the utterance *the circle above the green triangle* and the situation *bbcabgt*. With the $C'$ values learned

[2] There is no entry for *below* because it is not encountered in the examples.

for *circle*, *green* and *triangle*, the choice of $ab$ for *above* leads to a match for this utterance, because $ab(t_1, t_2)$ is supported by the situation, while $ab^t(t_1, t_2)$ is not. This approach relies on the correctness of the object identifications as established by the unary predicates. The words *to* and *of* occur only in the phrases *to the left of* and *to the right of*, which ensures that the binary predicate *le* is always in the situation when they occur. However, a similar attempt to assign a definite order of arguments to *le* will fail for these words.

*Other Languages, Other Phenomena.* We have gathered comparable samples for several other languages, which exhibit various other phenomena. For example, in our sample for Mandarin, a circle is designated as *yuan* or *yuan xing*, a triangle as *san jiao xing* and a square as *zheng fan xing*. The English utterance *the triangle below the circle* is rendered as *yuan xing xia mian de san jiao xing*. In this case, the denotation of the utterance (the triangle) is mentioned last in the utterance rather than first; this is a case in which the parameter *which* must be *last* rather than *first*.

Also in our sample for Mandarin, *san* and *jiao* always co-occur, and both of their $C'$ values are the unary predicate *tr*; analogously, *zheng* and *fan* both have the value *sq*. If two (or more) words always co-occur and have a non-empty meaning, there may be no evidence for which word should be assigned the meaning. In our sample, *tr* can be assigned to either *san* or *jiao* and the denotation function will be unaffected. [3]

Another phenomenon is present in our sample for Greek: the word for circle appears in three forms: *kyklos*, *kyklou* and *kyklo*, depending on whether it is the object of a preposition, and, if so, which preposition. A combination of morphological and semantic evidence would suggest that these three words are in fact one word. If, however, we treat them as separate words, in the case of *kyklou*, the binary predicate *le* will be present in every situation in which the word is used, so that its $C'$ value is $\{ci, le\}$ in the limit. Similarly, *kyklo* will always co-occur with the binary predicate $ab$. We would like a criterion to select one of the two possibilities.

## 4 The Learning Algorithm

Based on the ideas developed above, we propose a learning algorithm, and give a set of assumptions under which it finitely converges to a correct meaning function.

*Further Assumptions about $M$.* We make additional assumptions about the meaning transducer $M$. In Assumption 1, we have assumed that $\gamma(w)$ depends only on the input word $w$. If $W'$ is a set of words, we define

$$\gamma(W') = \{\gamma(w) : w \in W', \gamma(w) \neq \varepsilon\}.$$

[3] It would be preferable to recognize lexical items that are combinations of words.

We define the set of all utterances in $L(M)$ that contain $w$:

$$L_M(w) = \{u \in L(M) : w \in c(u)\}.$$

A word $w_1$ **implies** a word $w_2$, denoted $w_1 \rightarrow_M w_2$ if $L_M(w_1) \subseteq L_M(w_2)$; this is true if every utterance in $L(M)$ that contains $w_1$ also contains $w_2$. Two words $w_1$ and $w_2$ **always co-occur**, denoted $w_1 \leftrightarrow_M w_2$, if $L_M(w_2) = L_M(w_1)$. This is an equivalence relation; its equivalence classes are **co-occurrence classes**. In our English example, the words *to* and *of* always co-occur and each one is implied by the word *left*. In our Mandarin example, the words *san* and *jiao* always co-occur.

To deal with co-occurrence classes instead of words as the units to which meanings are assigned, we assume that $\gamma$ is well-behaved with respect to co-occurrence classes. We say that $\gamma$ is **single-valued** if for every co-occurrence class $K$, $\gamma$ assigns a nonempty output to at most one word from $K$.

**Assumption 2.** *The output function $\gamma$ is single-valued and for any single-valued output function $\gamma'$ such that $\gamma(K) = \gamma'(K)$ for all co-occurrence classes $K$, $M(u) = M'(u)$ for every $u \in L(M)$, where $M'$ is $M$ with output function $\gamma'$.*

For example, in our sample of Mandarin: either *san* or *jiao* can be assigned the meaning *tr* without affecting the resulting values $M(u)$ for utterances $u \in L(M)$. This assumption is not true in our Greek example: if *o* rather than *kyklos* is assigned the output *ci*, then the output of *o mple kyklos pano apo to tetragono* is changed from $\langle bl, ci, ab, sq\rangle$ to $\langle ci, bl, ab, sq\rangle$. [4]

We next assume that the language of denoting utterances and their meanings are sufficient to determine the value of $\gamma$ for each co-occurrence class. Define for each co-occurrence class $K$,

$$P_M(K) = \bigcap \{c(M(u)) : u \in L(M), K \subseteq c(u)\}.$$

This is all predicates common to meanings of utterances from $L(M)$ that contain the words in $K$. Note that for all co-occurrence classes, $\gamma(K) \subseteq P_M(K)$. The following assumption strengthens this to equality.

**Assumption 3.** *For all co-occurrence classes $K$, $\gamma(K) = P_M(K)$.*

This assumption holds of the transducer $M_1$. For example, the value of $P_M$ for the co-occurrence class $\{of, to\}$, is $\emptyset$, witnessed by the utterances *the circle to the right of the square*, *the triangle to the left of the circle* and *the square to the right of the triangle* and their corresponding sets of predicates: $\{ci, le^t, sq\}$, $\{tr, le, ci\}$ and $\{sq, le^t, tr\}$. In the case of Greek there are occurrences of *kyklou* with both *sta deksia* and *sta aristera* that eliminate both *le* and $le^t$ from the value of $P_M$.

**Lemma 1.** *Under Assumptions 1, 2, and 3, knowing the set of co-occurrence classes $K$ and the values $P_M(K)$ is sufficient to compute the value of $\gamma(u)$ for every $u \in L(M)$.*

[4] The denotation function is unaffected; perhaps this assumption should be weakened.

(The proof of Lemma 1 is omitted for lack of space.)

However, in the setting we consider, what is observed is the primary versions of binary predicates. We therefore define a variant of $P_M(K)$ in which predicates are first transformed to their primary versions.

$$PP_M(K) = \bigcap \{ \mathit{primary}(c(M(u))) : u \in L(M), K \subseteq c(u) \}.$$

For the example of the transducer $M_1$ and the co-occurrence class $\{\mathit{of}, \mathit{to}\}$, the value of $PP_M$ is $\{\mathit{le}\}$ because whenever these two words occur in an utterance $u$ from $L(M)$, either *right* or *left* occurs, and therefore *le* occurs in $\mathit{primary}(c(M(u)))$. Similarly, in the case of Greek, the value of $PP_M$ for the class containing *kyklou* consists of *ci* and *le*. A useful property of $PP_M(K)$ is that it gives correct information about the unary predicates.

**Lemma 2.** *For every co-occurrence class $K$, $PP_M(K) \cap P = P_M(K) \cap P$.*

(The proof of Lemma 2 is omitted for lack of space.)

*The Learning Algorithm.* We assume that the learning algorithm receives examples $(S_i, u_i)$ for $i = 1, 2, \ldots$ and responds to each one by hypothesizing a meaning function $\gamma_n$ based on the first $n$ examples. The criterion of success is whether the algorithm finitely converges to a meaning function $\gamma'$ such that $\gamma(u) = \gamma'(u)$ for all utterances $u \in L(M)$.

After receiving the example $(S_n, u_n)$, the learner computes the intersection of the sets of predicates seen in every situation so far, as follows.

$$G_n = \bigcap_{i=1}^{n} \mathit{predicates}(S_i).$$

Let $G$ be the set of **background predicates**, that is, all predicates that occur in every situation $S_i$.

$$G = \bigcap_{i} \mathit{predicates}(S_i).$$

Then $G_n$ finitely converges to $G$, because the set of predicates in any situation is finite.

The algorithm maintains a partition $\mathcal{K}_n$ of the words it has seen, in which two words $w_1$ and $w_2$ are in the same class if they occur in exactly the same set of utterances $u_i$ with $1 \leq i \leq n$. For each class $K \in \mathcal{K}_n$, the learning algorithm computes the set of unary predicates that occur in every situation $S_i$ for which the utterance $u_i$ contains the class $K$:

$$U_n(K) = P \cap \bigcap \{ \mathit{predicates}(S_i) : 1 \leq i \leq n, K \subseteq c(u_i) \}.$$

The algorithm uses these sets to define a **partial meaning function** $g_n$ as follows. For each class $K \in \mathcal{K}_n$, if $(U_n(K) - G_n)$ is nonempty then the algorithm selects one word $w \in K$ and one predicate $p \in (U_n(K) - G_n)$ and defines $g_n(w) = p$. For all other words, the algorithm defines $g_n(w) = \varepsilon$.

The map $g_n$ translates any utterance into a sequence of unary predicates. For example, using the map $g_n$ derived from the data in Sect. 3, the translation of *the green circle to the right of the red triangle* is $\langle \mathit{gr}, \mathit{ci}, \mathit{re}, \mathit{tr} \rangle$.

*Resolving Argument Order.* The partial meaning function $g_n$ is used to try to gather information about the possible orders of arguments of binary predicates as follows. Let $u$ be a denoting utterance in a situation $S$, with partial translation $g_n(u) = \langle p_1, p_2, \ldots, p_k \rangle$. Let $\langle t_{i_1}, t_{i_2}, \ldots, t_{i_r} \rangle$ be a finite sequence of distinct things from the situation $S$. We say that this sequence is **compatible** with the partial translation $g_n(u)$ if there exists a partition of the sequence $\langle 1, 2, \ldots, k \rangle$ into $r$ (possibly empty) non-overlapping consecutive intervals $I_1, I_2, \ldots, I_r$ such that $p_\ell(t_{i_j})$ is supported by $S$ for every $j = 1, \ldots, r$ and $\ell \in I_j$.

We define the set of possible binary predicates $\mathit{possible}(S, u)$ as follows. For each atom $r(t_i, t_j)$ in $S$, $r$ is included in $\mathit{possible}(S, u)$ if there is an ordering compatible with $g(u)$ in which $t_i$ immediately precedes $t_j$, and $r^t$ is included in $\mathit{possible}(S, u)$ if there is an ordering compatible with $g(u)$ in which $t_j$ immediately precedes $t_i$. Note that

$$\mathit{primary}(\mathit{possible}(S, u)) \subseteq R \cap \mathit{predicates}(S).$$

For example, if the situation $S$ is a big red triangle ($t_1$) to the left of a big green circle ($t_2$), with a big red square ($t_3$) below the circle, then the only orderings compatible with $\langle gr, ci, re, tr \rangle$ are $\langle t_2, t_1 \rangle$, $\langle t_3, t_2, t_1 \rangle$, $\langle t_2, t_3, t_1 \rangle$ and $\langle t_2, t_1, t_3 \rangle$. Then in the computation of $\mathit{possible}(S, u)$, $le^t$ is included (because $le(t_1, t_2)$ is in the situation and $t_2$ immediately precedes $t_1$ in some compatible ordering) but $le$ is not (because $t_1$ does not immediately precede $t_2$ in any compatible ordering.) Considering the occurrences of $t_2$ and $t_3$ in the compatible orderings, both $ab$ and $ab^t$ will be included.

The learner defines for each class $K \in \mathcal{K}_n$ a set of binary predicates as follows.

$$B_n(K) = \bigcap \{\mathit{possible}(S_i, u_i) : 1 \leq i \leq n, K \subseteq c(u_i)\}.$$

Finally, the learner uses $B_n(K)$ to extend $g_n$ to a hypothesized meaning function $\gamma_n$ as follows. For each $w$ such that $g_n(w) \neq \varepsilon$, $\gamma_n(w)$ is set to $g_n(w)$. For each class $K \in \mathcal{K}_n$, if $(U_n(K) - G_n) = \emptyset$ and $(B_n(K) - G_n) \neq \emptyset$, then a word $w$ is selected from $K$ and a predicate $q$ from $(B_n(K) - G_n)$ and $\gamma_n(w)$ is set to $q$. For all remaining words $w$, $\gamma_n(w)$ is set to $\varepsilon$. This concludes the description of the learning algorithm.

We note that the algorithm prefers to assign a unary predicate as the meaning of a co-occurrence class $K$ if possible. This means that it prefers $ci$ to $le$ as the meaning of the co-occurrence class of *kyklou* in our Greek sample.

*Correctness of the Algorithm.* We make some additional assumptions about the sequence of examples $(S_i, u_i)$ and then prove the following.

**Theorem 1.** *Under Assumptions 1 through 6, the learning algorithm finitely converges to a meaning function $\gamma'$ such that $\gamma'(u) = \gamma(u)$ for every $u \in L(M)$.*

For the following assumptions, we consider only words $w$ and co-occurrence classes $K$ that actually appear in some example $(S_i, u_i)$. Our first assumption about the data sequence $(S_i, u_i)$ guarantees that the partition $\mathcal{K}_n$ finitely converges to the correct co-occurrence classes for $L(M)$.

**Assumption 4.** *For all pairs of words $w_1$ and $w_2$, $w_1 \leftrightarrow_M w_2$ if and only if for all $i$, $c(u_i)$ contains both $w_1$ and $w_2$ or neither $w_1$ nor $w_2$.*

The second assumption about the sequence of examples allows the algorithm to compute $PP_M(K)$ in the limit. If $K$ is a co-occurrence class, let

$$C(K) = \bigcap_i \{\mathit{predicates}(S_i) : K \subseteq c(u_i)\},$$

and $C'(K) = C(K) - G$.

**Assumption 5.** *For each co-occurrence class $K$, $C'(K) = PP_M(K)$.*

Note that this implies that no background predicate is in $\gamma(K)$ for a co-occurrence class $K$ that appears in some example $(S_i, u_i)$. Our final assumption regarding the data is that if the unary predicates are learned correctly, then compatibility considerations are enough to rule out any incorrect binary predicates.

**Assumption 6.** *Suppose $g$ is a partial meaning function such that for all co-occurrence classes $K$, $g(K) = \gamma(K)$ if $\gamma(K)$ is a unary predicate and $g(K) = \emptyset$ otherwise. Then for every co-occurrence class $K$ such that $\gamma(K)$ is not a unary predicate and every predicate $q \in (R \cup R^t - G)$, such that $q \notin \gamma(K)$, there exists an example $(S_i, u_i)$ such that $q \notin \mathit{possible}(S_i, u_i)$, where possible is computed with respect to $g$.*

Assumptions 4, 5, and 6 are satisfied by the data in Sect. 3 with respect to $M_1$.

*Proof (of Theorem 1).*

By Assumption 4, the partition $\mathcal{K}_n$ finitely converges to the correct co-occurrence classes of $L(M)$, so let $n$ be sufficiently large that this is true. Because $G_n$ finitely converges to the background predicates $G$ and $U_n(K)$ finitely converges to the unary predicates in $PP_M(K)$, we have that $(U_n(K) - G_n)$ finitely converges to the unary predicates in $PP_M(K)$, which by Lemma 2 and Assumption 3 are just the unary predicates in $\gamma(K)$. If $n$ is also sufficiently large that this is true, then $g_n(K) = \gamma(K)$ for all co-occurence classes $K$ such that $\gamma(K)$ is a unary predicate and $g_n(K) = \emptyset$ for all other co-occurrence classes $K$.

Thus $g_n$ satisfies the hypotheses of Assumption 6. Consider any co-occurrence class $K$ such that $\gamma(K)$ is not a unary predicate. If $\gamma(K) = \emptyset$, then for every binary predicate $q \in (R \cup R^t - G)$, there exists an example $(S_i, u_i)$ such that $q \notin \mathit{possible}(S_i, u_i)$. Thus, for $n$ sufficiently large, $(B_n(K) - G_n) = \emptyset$ and $\gamma_n(K) = \emptyset$.

Suppose $\gamma(K) = \{q\}$ for some $q \in R \cup R^t$. Then $q \notin G$, so $q \in (R \cup R^t - G)$, by Assumption 5. For all sufficiently large $n$, $q$ is in $\mathit{possible}(S_i)$ if $K \subseteq c(u_i)$ for all $1 \leq i \leq n$. This is true because $u_i$ is a denoting utterance in $S_i$, so there is a match $f$ from $x_1, \ldots, x_k$ to things in $S_i$ such that all the atoms of $\pi(M(u_i))$ are supported in $S_i$. Thus, there is an ordering $f(x_1), f(x_2), \ldots, f(x_k)$ of things from $S_i$ compatible with the subsequence of $M(u_i)$ consisting of unary predicates (which is equal to $g_n(u_i)$), and in which $q(t_j, t_{j+1})$ for some $j$. Thus, $q \in (B_n(K) - G_n)$ for all sufficiently large $n$. Every other predicate from $(R \cup R^t - G)$ is eliminated by some example $(S_i, u_i)$, by Assumption 6.

Thus, for sufficiently large $n$, $\gamma_n$ finitely converges to a meaning function $\gamma'$ such that $\gamma'(K) = \gamma(K)$ for all co-occurrence classes $K$ of $L(M)$. By Assumption 2, $\gamma'(u) = \gamma(u)$ for all utterances $u \in L(M)$. $\square$

## 5 Discussion and Future Work

What about computational feasibility? Word co-occurrence classes, the sets of predicates that have occurred with them, and background predicates can all be maintained efficiently and incrementally. However, the problem of determining whether there is a match of $\pi(M(u))$ in a situation $S$ when there are $N$ variables and at least $N$ things, includes as a special case finding a directed path of length $N$ in the situation graph, which is NP-hard in general. Also, our method of determining the order of arguments of binary predicates potentially involves considering all possible orderings of the distinct things in a situation. It is likely that human learners do not cope well with situations involving arbitrarily many things, and it is important to find good models of focus of attention.

Our model suggests that learning meaning not only facilitates learning syntax, but also precedes it. We agree with Tellier's suggestion [11] that "the acquisition of a conceptual representation of the world is necessary *before* the acquisition of the syntax of a natural language can start."

We have tested our model in the simple domain of geometric shapes with sets of utterances in a number of natural languages, including Arabic, English, Greek, Hebrew, Hindi, Mandarin, Russian, Spanish and Turkish [16]. These experiments show the robustness of our assumptions for this domain and the adequacy of our model to deal with crosslinguistic data.

Further work is required to relax some of the more restrictive assumptions. For example, in the current framework, disjunctive meanings cannot be learned, nor can a function that assigns meanings to more than one of a set of co-occurring words. Statistical approaches may produce more powerful versions of the models we consider. We plan to develop our model to incorporate production and syntax learning by the learner, as well as corrections and expansions from the teacher.

The authors thank Ronny Dakdouk, Kevin Gold, Melis Inan, Gaja Jarosz, Edo Liberty, Lev Reyzin, Brian Scassellati, Nikhil Srivastava, Antonis Stampoulis, and Yinghua Wu for their help with aspects of this work.

## References

1. Feldman, J.: Some decidability results on grammatical inference and complexity. Information and Control 20, 244–262 (1972)
2. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
3. Sakakibara, Y.: Recent advances of grammatical inference. Theoretical Computer Science 185, 15–45 (1997)
4. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition 38, 1332–1348 (2005)

5. Feldman, J.: Real language learning. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 114–125. Springer, Heidelberg (1998)
6. Anderson, J.: Induction of augmented transition networks. Cognitive Science 1, 125–157 (1977)
7. Hill, J.A.C.: A Computational Model of Language Acquisition in the Two-year-old. Indiana University Linguistics Club, Indiana (1983)
8. Hamburger, H., Wexler, K.: A mathematical theory of learning transformational grammar. Journal of Mathematical Psychology 12, 137–177 (1975)
9. Schlesinger, I.: Production of utterances and language acquisition. In: Slobin, D. (ed.) The Ontogenesis of Grammar, pp. 63–103. Academic Press, New York (1971)
10. Schaerlaekens, A.M.: The two-word sentence in child language development. Mouton, The Hague (1973)
11. Tellier, I.: Meaning helps learning syntax. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 25–36. Springer, Heidelberg (1998)
12. Oates, T., Armstrong, T., Harris, J., Nejman, M.: On the relationship between lexical semantics and syntax for the inference of context-free grammars. In: AAAI, pp. 431–436 (2004)
13. Gold, K., Scassellati, B.: A robot that uses existing vocabulary to infer non-visual word meanings from observation. In: AAAI, pp. 883–888 (2007)
14. Brown, R., Bellugi, U.: Three processes in the child's acquisition of syntax. Harvard Educational Review 34, 133–151 (1964)
15. Brown, R., Fraser, C.: The acquisition of syntax. In: Cofer, C., Musgrave, B. (eds.) Verbal behavior and learning: Problems and processes, pp. 158–197. McGraw-Hill, New York (1963)
16. Angluin, D., Becerra-Bonache, L.: Learning meaning before syntax. Technical Report YALE/DCS/TR1407, Computer Science Department, Yale University (2008)
17. Siskind, J.: A computational study of cross-situational techniques for learning word-to-meaning mappings. Cognition 61, 39–61 (1996)
18. Smith, K., Smith, A.D.M., Blythe, R.A., Vogt, P.: Cross-situational learning: a mathematical approach. In: Vogt, P., Sugita, Y., Tuci, E., Nehaniv, C.L. (eds.) EELC 2006. LNCS (LNAI), vol. 4211, pp. 31–44. Springer, Heidelberg (2006)
19. Thompson, C.A., Mooney, R.J.: Acquiring word-meaning mappings for natural language interfaces. Journal of Artificial Intelligence Research 18, 1–44 (2003)

# Schema-Guided Induction of Monadic Queries

Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren

University of Lille and INRIA Lille-Nord Europe, France

**Abstract.** The induction of monadic node selecting queries from partially annotated XML-trees is a key task in Web information extraction. We show how to integrate schema guidance into an RPNI-based learning algorithm, in which monadic queries are represented by pruning node selecting tree transducers. We present experimental results on schema guidance by the DTD of HTML.

## 1 Introduction

Various machine learning techniques have been applied for automating Web information extraction. These range from classification [11,12,16], conditional random fields [14], inductive logic programming [7], to tree automata induction [19,13,4,15].

We study information extraction from well-structured HTML documents generated by some database. The basic problem is to find monadic queries that select informative nodes in unranked trees. Surprisingly, no schema information has been taken into account so far, even not the document type definition (DTD) of HTML. Inferred DTDs obtained from some independent algorithm have not been exploited either [1]. Instead, all available techniques rely on some finite set of attributes or local properties of the environment of nodes in the trees. The reason for ignoring schema information may be that it cannot be integrated into most approaches. Tree automata based techniques for the inference of regular tree languages are the exception [4,15], as we show in this article, but it requires considerable effort. Automata for local tree languages are not sufficient [19,13].

In this article, we introduce schema guidance into the learning algorithm for monadic queries represented by pruning node selecting tree transducers (pNSTTs) presented in [4]. These are tree automata that recognize monadic queries represented as tree languages. The first idea is to learn only such queries that are consistent with the schema, in that they select nodes in trees satisfying the schema only. This is known as *domain bias* [9] in the more restrictive framework for inference of regular word languages. The second idea is that schema information is useful in pruning heuristics for interactive query learning.

Checking the consistency of queries with respect to schemas amounts to testing language inclusion $L(A) \subseteq L(D)$ for stepwise tree automata $A$ which may be nondeterministic [8,5] and (deterministic) DTDs $D$ over the same signature $\Sigma$. This can be done in time $O(|A| * |\Sigma| * |D|)$ due to a recent algorithm [6], motivated by the application presented here, which is quite evolved. It avoids quadratic blowups in two places: in the translation of DTDs to bottom-up deterministic tree automata by introducing factorization, and by avoiding automata complementation all over (the completion of a binary tree automaton may be of quadratic size).

Interactions between the user and the system are essential for keeping the overall amount of annotations reasonably small. Pruning heuristics serve for interactive learning of pNSTT queries from partially annotated example trees. The only completeness assumption that can be maintained in interactive information extraction is that all selected ancestors of positively annotated nodes are annotated. pNSTTs restore complete annotations by pruning subtrees that do not contain positive annotations (and interpreting missing annotations above positive annotations as negative). Pruning means to replace subtrees by placeholder symbols. pNSTTs use a single symbol $\top$ that denotes the set of all possible trees. If the schema is defined by a deterministic tree automaton $S$ (or a DTD), we can refined this idea and replace subtrees by their type, which is the unique state into which it is evaluated by $S$. This leads to a generalization of pNSTTs to schema-dependant $S$-pNSTTs. Algorithmically, the RPNI algorithm [17] has to check whether a tree automaton is an $S$-pNSTT, i.e., whether it actually represents a valid query. We present a polynomial time algorithm for this purpose.

We have implemented the complete interactive learning algorithm for $S$-pNSTTs from scratch including the two aspects of schema guidance by $S$. This is done in such a way that we can run the same algorithm with or without schema consistency, schema-guided pruning, or a state typing heuristics. No other hidden heuristics or preprocessing steps have been used. A preliminary experimental evaluation yields the following insights on schema guidance by the DTD of HTML. First, it might be of interest to observe that the number of state merges is decreased considerably by schema guidance, while the learning time remains stable. This means that the time gained by fewer merge failures is sufficient to account for the additional inclusion tests. We didn't expect this effect at the beginning. It shows that the approach is feasible. Second, the overall learning quality (precision and recall) do not change very much. Typing heuristics permit to avoid wrong generalizations and allow to improve performance of learning algorithms. The effect of schema guidance remains questionable, while schema guided pruning works well. We only use HTML documents, thus the DTD of HTML. Clearly XML queries with other XML schemas must be considered to answer the remaining questions. In the interactive setting, typing heuristics and schema-guided pruning allow to decrease the number of user interactions needed to achieve a consistent query.

## 2 Schemas, Tree Automata, and Inclusion Checking

Schemas and node selection queries for unranked trees can be defined in various XML standards or by tree automata [8]. We will use DTDs as for the definition of HTML and stepwise tree automata for encoding DTDs, and defining queries.

```
<!ELEMENT doc
(block+)> <!ELEMENT block (text,(link,text?)?
                    |link,text?)>
<!ELEMENT text
(#PCDATA)> <!ELEMENT link
(#PCDATA)>
```

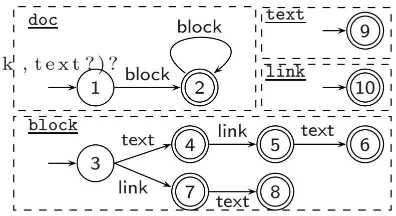


**Fig. 1.** An example DTD and the corresponding Glushkov automata

We recall how to translate DTDs into deterministic stepwise tree automata, and discuss inclusion checking.

Let $\Sigma$ be a finite set, $\mathbb{N}$ the set of natural numbers (starting from 1) and $\mathbb{B} = \{0, 1\}$ the set of Booleans. The set of unranked trees $T_\Sigma$ is the least set that contains all tuples $a(t_1, \ldots, t_n)$ where $a \in \Sigma$, $n \geq 0$, and $t_1, \ldots, t_n \in T_\Sigma$. A node of a tree is a word $\pi \in \mathbb{N}^*$ (using classical Dewey encoding). We write $\epsilon$ for the empty word and $i \cdot \pi$ for the concatenation of letter $i$ with word $\pi$. The set of nodes of an unranked tree is $\mathsf{nodes}(a(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i \cdot \pi \mid 1 \leq i \leq n,\ \pi \in \mathsf{nodes}(t_i)\}$. The label of a tree $t$ at a node $\pi \in \mathsf{nodes}(t)$ is denoted by $t(\pi) \in \Sigma$, the root of $t$ is distinguished by $\mathsf{root}(t)$. By $t_{|\pi}$, we denote the subtree of $t$ rooted by $t(\pi)$.

A *schema* over $\Sigma$ is a regular language of unranked trees $L \subseteq T_\Sigma$. We will use two kinds of schema definitions: XML DTDs and stepwise tree automata [5], possibly with factorization [6] in order to avoid a quadratic blowup when encoding DTDs.

A DTD $D$ over $\Sigma$ consists of a collection of one-unambiguous regular expressions $(e_a^D)_{a \in \Sigma}$ [3], and a (unique) accepting symbol $\mathsf{start}_D \in \Sigma$. The word language $L(e) \subseteq \Sigma^*$ defined by a regular expression $e$ over $\Sigma$ is defined as usual. For every DTD $D$ we define tree languages $L_a(D) \subseteq T_\Sigma$ by the least solution of the following system of equations where $a, a_1, \ldots, a_n \in \Sigma$:

$$L_a(D) = \{a(t_1, \ldots, t_n) \mid a_1 \cdots a_n \in L(e_a^D), a_i = \mathsf{root}(t_i), t_i \in L_{a_i}(D), 1 \leq i \leq n\}$$

The above definition basically means that the word obtained by concatenating the labels of the children of each node labeled by $a$ must be in $L(e_a^D)$. The language of the schema is $L(D) = L_{\mathsf{start}_D}(D)$. An example DTD is given in Fig. 1. The regular expressions of DTDs can be converted into finite automata recognizing the same language by Glushkov's construction [2]. These automata are deterministic, since the regular expressions in DTDs are one-unambiguous [3]. The size of the Glushkov automaton $G_e$ of a regular expression $e$ is at most $|\Sigma| * |e|$, which is the maximal number of transitions in deterministic automata over $\Sigma$ with $|e|$ states.

A *stepwise tree automaton* $A$ over $\Sigma$ is a standard tree automaton over the ranked signature $\Sigma_@ = \Sigma \uplus \{@\}$, where all elements of $\Sigma$ are constants and @ is a binary function symbol. The rules of $A$, denoted $\mathsf{rules}(A)$, are of the form $a \rightarrow q$, $q_1 @ q_2 \rightarrow q$, or $q_1 \xrightarrow{\epsilon} q_2$, where $a \in \Sigma$ and $q, q_1, q_2 \in \mathsf{states}(A)$, the set of states of $A$. Automaton $A$ is (bottom-up) *deterministic*, if it has no $\epsilon$-rules and

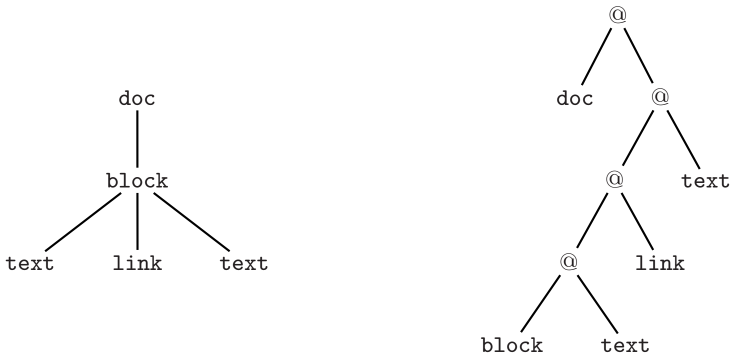


**Fig. 2.** Currying the unranked tree `doc`(`block`(`text`,`link`,`text`) into the binary tree `doc`@(`block`@`text`@`link`@`text`)

no two rules with the same left-hand side. We call an automaton *productive* if all of its states are accessible and co-accessible.

Let $T^{\text{bin}}_{\Sigma_@}$ be the set of binary trees over the signature $\Sigma_@$ and $L^{\text{bin}}(A) \subseteq T_{\Sigma_@}$ be the set of binary trees recognized by $A$. Every unranked tree $t \in T_\Sigma$ can be encoded *via* Currying into a binary tree in $T_{\Sigma_@}$, so that $\texttt{curry}(a) = a$ and if $n \geq 1$ then $\texttt{curry}(a(t_1, \ldots, t_n)) = \texttt{curry}(a(t_1, \ldots, t_{n-1}))@\texttt{curry}(t_n)$. An example is depicted in Fig. 2. Here, we write $t_1@t_2$ instead of $@(t_1, t_2)$. The language $L(A) \subseteq T_\Sigma$ of unranked trees recognized by a stepwise tree automaton $A$ is the set:

$$L(A) = \{t \in T_\Sigma \mid \texttt{curry}(t) \in L^{\text{bin}}(A)\}$$

The direct transformation of DTDs into deterministic stepwise tree automata implies a quadratic blowup in the size of the DTD. To avoid such a blowup, we introduce *factorized tree automata* [6]. These are stepwise tree automata with $\epsilon$-rules which represent stepwise tree automata in a compact manner. Nevertheless, we can still define an appropriate notion of determinism for factorized tree automata in order to deal with DTDs.

**Definition 1.** *A factorized tree automaton $F$ over $\Sigma$ is a stepwise tree automaton over $\Sigma$ with $\epsilon$-rules, and a partition into two sorts* $\mathsf{states}(F) = \mathsf{states}_1(F) \uplus \mathsf{states}_2(F)$ *such that if $q_1@q_2 \to q$ in* $\mathsf{rules}(F)$ *then $q_1 \in$* $\mathsf{states}_1(F)$ *and $q_2 \in$* $\mathsf{states}_2(F)$. *$F$ is (bottom-up)* deterministic *if its $\epsilon$-free part is (bottom-up) deterministic and all $q \in$* $\mathsf{states}(F)$ *have at most one outgoing $\epsilon$-edge, the target of which must be of the other sort.*

It should be noticed that the idea of factorization is equally provided by the tree automata for unranked trees proposed in [18].

The collection of Glushkov automata $(G_a)_{a\in\Sigma}$ of a DTD $D$ can now be translated in linear time to a factorized tree automaton $F$ with $\mathsf{states}_1(F) = \uplus_{a\in\Sigma}\mathsf{states}(G_a)$ and $\mathsf{states}_2(F) = \Sigma$. The rules of $F$ are defined as follows:

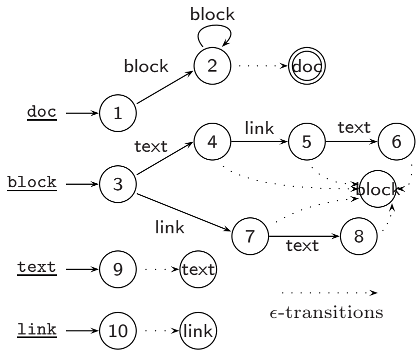


**Fig. 3.** The deterministic factorized tree automaton for the DTD in Fig. 1

$q_1 \xrightarrow{a} q_2 \in \mathsf{rules}(G_a)$ iff $q_1@a \to q_2 \in \mathsf{rules}(F)$, and $q \in \mathsf{final}(G_a)$ iff $q \xrightarrow{\epsilon} a \in \mathsf{rules}(F)$. This correspondence is equally useful for drawing factorized tree automata as in Fig. 3. The single final state of $F$ is the accepting label of $D$, i.e., $\mathsf{final}(F) = \{\mathsf{start}_D\}$. Indeed, $L(A) = L(F)$. Furthermore, $F$ is deterministic as a factorized tree automaton. Its $\epsilon$-free part is deterministic, since all Glushkov automata are deterministic. The only states having outgoing $\epsilon$-edges are the final states of Glushkov automata, which are of sort 1. They have at most one outgoing $\epsilon$-edge, since every $q$ belongs to at most one Glushkov automaton $G_a$; the target of this edge is of other sort 2. In principle, these $\epsilon$-edges can be eliminated in order to obtain a deterministic tree automaton, but this could lead to a quadratic size increase.

**Theorem 1 ([6]).** *Language inclusion $L(A) \subseteq L(F)$ between stepwise tree automata $A$ with $\epsilon$-rules and deterministic factorized tree automata $F$ can be tested in time $O(|A| * |F|)$.*

The most important point here is that one does not have to compute the automaton for the complement of $F$, which could grow up quadratically to $O(|F|^2)$ since completion is necessary before swapping final states. The second point is that factorization avoids a quadratic blowup when translating DTDs into stepwise tree automata. As a corollary, we can check language inclusion between stepwise tree automata $A$ and DTDs $D$ over signature $\Sigma$ in time $O(|A| * |\Sigma| * |D|)$. Third, the inclusion test is incremental with respect to adding $\epsilon$-edges to $A$. This can be used to check inclusion incrementally in a learning algorithm because, after state merging in $A$, we simply add back and forth $\epsilon$-edges rather than physically identifying the merged states.

## 3 Schema-Guided pNSTTs for Monadic Queries

We generalize the notion of pNSTTs to $S$-pNSTTs such that pruning is guided by a schema $S$, and present a polynomial time algorithm testing whether a deterministic tree automaton is an $S$-pNSTT.

**Node Selecting Tree Transducers.** A *monadic query* $Q$ in unranked trees over $\Sigma$ is a total function mapping trees $t \in T_\Sigma$ to sets of nodes $Q(t) \subseteq \mathsf{nodes}(t)$. We call a monadic query $Q$ *consistent with a schema* $L \subseteq T_\Sigma$ if it selects nodes only in trees satisfying the schema, i.e., if $Q(t) = \emptyset$ for all $t \in T_\Sigma \setminus L$.

A (Boolean) *annotated tree* over $\Sigma$ is a tree over $\Gamma = \Sigma \times \mathbb{B}$. Every annotated tree $s \in T_\Gamma$ can be decomposed in a unique manner into two trees $t \in T_\Sigma$ and $\beta \in T_\mathbb{B}$ with the same sets of nodes $\mathsf{nodes}(s) = \mathsf{nodes}(t) = \mathsf{nodes}(\beta)$ such that for all nodes $\pi$ therein, $s(\pi) = (t(\pi), \beta(\pi))$. In this case, we write $s = t * \beta$.

A language $L \subseteq T_\Gamma$ of annotated trees defines a relation $r_L \subseteq T_\Sigma \times T_\mathbb{B}$ between trees of the same structure, which is $r_L = \{(t, \beta) \mid t * \beta \in L\}$. We call $L$ *functional* if this relation $r_L$ is a partial function. In other words, for every tree $t \in T_\Sigma$ there exists at most one tree $\beta \in T_\mathbb{B}$ such that $t * \beta \in L$. A *node selecting tree transducer (NSTT)* for $\Sigma$ is an automaton over $\Gamma$ which recognizes a functional language of annotated trees.

A completely annotated example for a query $Q$ is a tree $t * \beta$ where $\beta(\pi) = 1$ for all selected nodes $\pi \in Q(t)$ and $\beta(\pi) = 0$ otherwise. An algorithm for testing functionality and an RPNI algorithm for learning NSTTs from completely annotated examples have been presented in [4].

**Schema-Guided Pruning.** In Web information extraction, however, only partially annotated examples for the target query $Q$ are available. These are triples $(t, e_+, e_-)$ such that $e_+ \subseteq Q(t)$ and $e_- \subseteq \mathsf{nodes}(t) \setminus Q(t)$. The only completeness assumption for partially annotated examples that can be maintained is that selected nodes on paths from the root to some annotated selected node in $e_+$ are annotated, too. If $\pi \in e_+$ and $\pi' \in Q(t)$ is a prefix of $\pi$ then $\pi' \in e_+$.

Pruning is a method by which to deal with partially annotated examples. The idea is to cut down all subtrees from $t$ that do not contain nodes in $e_+$. These subtrees are replaced by special symbols, which indicate the type of the subtree. Then all other nodes are annotated by using the given partial annotation and the completeness assumption, i.e., a node is annotated by 1 if it is in $e_+$, and by 0 otherwise. The pNSTTs from [4] permit only a single type satisfied by all trees. When having schema information available, we can refine this approach by using the states of the schema as type information. This leads to the following formal definitions.

Let $S$ be a tree automaton over $\Sigma$ which defines the schema, and let us consider $\mathsf{states}(S)$ as symbols of arity 0. An *S-pruned annotated tree* is a tree $s \in T_{\Gamma \cup \mathsf{states}(S)}$. We call a tree $s \in T_{\Gamma \cup \mathsf{states}(S)}$ an *S-consistent pruning* of an annotated tree $t * \beta \in T_\Gamma$ if:

(i) $s(\pi) = t * \beta(\pi)$ if $s(\pi) \in \Gamma$, or
(ii) $t_{|\pi} \in L_{s(\pi)}(S)$ if $s(\pi) \in \mathsf{states}(S)$.

Now, let us consider a language $L \subseteq T_{\Gamma \cup \mathsf{states}(S)}$. $L$ is *S-cut-functional* if for all $s_1$ and $s_2$ in $T_{\Gamma \cup \mathsf{states}(S)}$ such that there exist $t*\beta_1$ and $t*\beta_2$ in $L$ with $s_1$, resp $s_2$, an $S$-consistent pruning of $t * \beta_1$, resp. $t * \beta_2$, then for all nodes $\pi \in \mathsf{nodes}(\beta_1) \cap \mathsf{nodes}(\beta_2)$, it holds that $\beta_1(\pi) = \beta_2(\pi)$. In other words, two $S$-consistent prunings of an annotated tree can not define contradictory annotations. We can now define $S$-guided pruning node selecting tree transducers.

**Definition 2.** *Given a schema $S$, an $S$-*pNSTT *over $\Sigma$ is a tree automaton over signature $(\Sigma \times \mathbb{B}) \cup \mathsf{states}(S)$ whose language is $S$-cut-functional.*

If $S_\top$ is the tree automaton with a unique state $\top$ that recognizes all unranked trees, then $S_\top$-pNSTTs coincide with pNSTTs presented before in [4]. Such pNSTTs can be learned by the variant pRNPI of RPNI, which tests for cut-functionality after all deterministic merges. In order to generalize pRNPI with schema-guided pruning, we need an algorithm testing $S$-cut-functionality for languages recognized by tree automata over the signature $\Gamma \cup \mathsf{states}(S)$.

Let $S$ be a deterministic factorized tree automaton over $\Sigma$ and let $A$ be a deterministic stepwise tree automaton over $\Gamma \cup \mathsf{states}(S)$. Deciding whether $A$ is an $S$-pNSTT, i.e., verifying that the language of $A$ is $S$-cut-functional, can be done by a ground Datalog program of polynomial size, and thus in polynomial time (see, e.g., [10]). Such a program can be inferred from $A$ and $S$ as described in the following.

The predicate $\mathtt{schema}(p,q)$ holds for $p \in \mathsf{states}(A)$ and $q \in \mathsf{states}(S)$ if there exist an annotated tree $t * \beta$ and an $S$-consistent pruning $s$ of $t * \beta$ such that $A$ evaluates $s$ to $p$ and $S$ evaluates $t$ to $q$. Note that only the second rule is concerned by pruning (state $q$ of $S$ is a constant symbol for $A$).

$$\frac{(a,b) \to p \in \mathsf{rules}(A) \qquad a \to q \in \mathsf{rules}(S)}{\mathtt{schema}(p,q).} \qquad \frac{q \in \mathsf{states}(S) \qquad q \to p \in \mathsf{rules}(A)}{\mathtt{schema}(p,q).}$$

$$\frac{p_1 @ p_2 \to p \in \mathsf{rules}(A) \qquad q_1 @ q_2 \to q \in \mathsf{rules}(S)}{\mathtt{schema}(p,q) :- \mathtt{schema}(p_1,q_1), \mathtt{schema}(p_2,q_2).} \qquad \frac{q \xrightarrow{\epsilon} q' \in \mathsf{rules}(S)}{\mathtt{schema}(p,q') :- \mathtt{schema}(p,q).}$$

The predicate $\mathtt{sim}(p,p')$ (for similar pruning) holds for two states $p$, $p'$ of $\mathsf{states}(A)$ if there exist two $S$-consistent-pruning $s$, $s'$ of the same annotated tree $t * \beta$ which are evaluated to $p$ and $p'$ by $A$. Here, only the second rule is directly concerned by pruning.

$$\frac{p \in \mathsf{states}(A)}{\mathtt{sim}(p,p).} \qquad \frac{q \in \mathsf{states}(S) \qquad q \to p' \in \mathsf{rules}(A)}{\mathtt{sim}(p,p') :- \mathtt{schema}(p,q).}$$

$$\frac{p_1 @ p_2 \to p \in \mathsf{rules}(A) \qquad p'_1 @ p'_2 \to p' \in \mathsf{rules}(A)}{\mathtt{sim}(p,p') :- \mathtt{sim}(p_1,p'_1), \mathtt{sim}(p_2,p'_2).}$$

The predicate $\mathtt{dast}(p,p')$ (different annotations on same tree) holds for two states $p$, $p'$ of $\mathsf{states}(A)$ if there exist an $S$-consistent pruning $s$ of $t * \beta$ and an $S$-consistent pruning $s'$ of $t * \beta'$ with a position $\pi$ verifying $s(\pi) \in \Gamma$, $s'(\pi) \in \Gamma$ and $s(\pi) \neq s'(\pi)$, such that $A$ evaluates $s$ to $p$ and $s'$ to $p'$. This predicate allows to detect failure for testing $S$-cut-functionality.

$$\frac{(a,b) \to p \in \mathsf{rules}(A) \qquad (a,\neg b) \to p' \in \mathsf{rules}(A)}{\mathtt{dast}(p,p').}$$

$$\frac{p_1@p_2 \to p \in \mathsf{rules}(A) \qquad p_1'@p_2' \to p' \in \mathsf{rules}(A)}{\begin{array}{c}\mathtt{dast}(p,p') :\!-\ \mathtt{dast}(p_1,p_1'), \mathtt{dast}(p_2,p_2').\\ \mathtt{dast}(p,p') :\!-\ \mathtt{dast}(p_1,p_1'), \mathtt{sim}(p_2,p_2').\\ \mathtt{dast}(p,p') :\!-\ \mathtt{sim}(p_1,p_1'), \mathtt{dast}(p_2,p_2').\end{array}}$$

The next proposition indicates how to determine whether an automaton is $S$-cut-functional by using the inferred Datalog program.

**Proposition 1.** *A deterministic tree automaton $A$ over $\Gamma \cup \mathsf{states}(S)$ is $S$-cut-functional with respect to a productive deterministic factorized tree automaton $S$ over $\Sigma$ if and only if there are no two states $p$, $p' \in \mathsf{states}(A)$ such that $\mathtt{dast}(p,p')$ holds, and either $p$, $p' \in \mathsf{final}(A)$ or $\mathtt{sim}(p,p')$ holds.*

Note that when defining the schema by automaton $S_\top$, predicate `schema` becomes trivial, and the test for $S_\top$-cut-functionality coincides with the cut-functionality test for pNSTTs presented in [4].

## 4 Schema-Guided Learning

We present the learning algorithm $\mathtt{RPNI}^{S,\mathsf{type}}_{\mathsf{prune},\mathsf{cons}}$ in Fig. 4 which is a variant of RPNI [17] that learns $S$-pNSTTs in a schema-guided manner. It is parameterized by a deterministic factorized tree automaton $S$ over $\Sigma$ which defines the schema. It inputs a finite set of completely annotated examples $E \subseteq T_{\Sigma\times\mathbb{B}}$ and a single partially annotated example $\langle t, e_+, e_-\rangle$ in $T_\Sigma \times \mathsf{nodes}(t)^2$. The algorithm could easily be extended to a set of partially annotated examples, but a single one is enough in most interactive learning scenarios.

The schema $S$ intervenes in the definition of the pruning algorithm $\mathsf{prune}_S$, in definitions of queries by deterministic $S$-pNSTTs over $\Sigma$ (distinguished by $S$-cut-functionality), and in consistency checking of queries with respect to the schema $L(S)$, which amounts to check for language inclusion (in polynomial time since $S$ is deterministic). We will consider several variants of the algorithm: whether $S$-guided pruning is done; whether $S$-consistency is checked for queries, and whether typing heuristics are used.

Learning without schema means to choose $S = S_\top$, the automaton with a single state that accepts all trees. $S$-consistency checking for queries can be switched on by choosing parameter $\mathsf{cons} =$ yes. Learning without pruning amounts to set $\mathsf{prune}_S$ to the identity function on annotated trees, i.e., $\mathsf{prune}_S(s) = s$ for all $s \in T_{\Sigma\times\mathbb{B}}$. With pruning, the function $\mathsf{prune}_S$ replaces subtrees, in which no nodes are selected, by their state with respect to $S$. This can be defined as follows where $t \in T_\Sigma$, $\beta \in T_\mathbb{B}$, $s, s_1, \ldots, s_n \in T_{\Sigma\times\mathbb{B}}$, $a \in \Sigma$, and $b \in \mathbb{B}$:

$$\begin{array}{ll}\mathsf{prune}_S(t * \beta) = \mathsf{eval}_S(t) & \text{if } \beta \in T_{\{0\}}\\ \mathsf{prune}_S((a,b)(s_1,\ldots,s_n)) = (a,b)(\mathsf{prune}_S(s_1),\ldots,\mathsf{prune}_S(s_n)) & \text{otherwise}\end{array}$$

Here $\mathsf{eval}_S(t) \in \mathsf{states}(S)$ is the state into which $S$ evaluates $t$. This state exists for all annotated examples since these are supposed to satisfy schema $S$. It is unique since $S$ is assumed deterministic.

```
$\mathtt{RPNI}^{S,\mathsf{type}}_{\mathsf{prune,cons}}$ ($E$, $\langle t, e_+, e_- \rangle$)
// sample of completely annotated examples $E \subseteq T_{\Sigma\times\mathbb{B}}$
// partially annotated example $\langle t, e_+, e_- \rangle \in T_\Sigma \times \mathsf{nodes}(t)^2$
// schema defined by a deterministic factorized tree automaton $S$ over $\Sigma$

// prune all example trees w.r.t. schema definition $S$//
let $E' = \{\mathsf{prune}_S(t' * \beta) \mid t' * \beta \in E\} \cup \{\mathsf{prune}_S(t * p_+)\}$
// compute the initial automaton
let $A$ be a deterministic $S$-pNSTT such that $L(A) = E'$
let states($A$)= $\{q_1, \dots, q_n\}$ in some admissible order
// generalize $A$ by state merging //
for $i = 1$ to $n$ do
    for $j = 1$ to $i - 1$ with $\mathsf{type}$ $(q_j)$=$\mathsf{type}$ $(q_i)$ do
        let $A'$ = det-merge($A, q_i, q_j$)
        if $A'$ is $S$ cut-functional // $S$-consistency of annotations on pruned trees
        and if $\mathsf{cons}$ =yes then $\{t \mid t * \beta \in L(A')\} \subseteq L(S)$ // query $S$-consistent
        and $A'$ consistent with sample $E$ and example $\langle t, e_+, e_- \rangle$
        then $A \leftarrow A'$
        else skip
Output : $A$
```

**Fig. 4.** Learning from completely and partially annotated example trees

State typing heuristics forbid to merge states of the automaton that have different types $\mathsf{type}(q_j) \neq \mathsf{type}(q_i)$. This is called typing bias in [9]. The definition of types depends on the application. In our algorithm, they are introduced by parameter $\mathsf{type}$. When no typing heuristics are used, we set $\mathsf{type}$ to a constant function on states. Otherwise, we say that a state $q$ of stepwise tree automaton $A$ has type $\mathsf{type}(q) = a \in \Sigma$, if $q$ is reachable from $a$ in the graph representing the stepwise automaton. We impose that no two states may have the same type, so that the graphical representation of $A$ can be decomposed into a disjoint union of independent connected components for all letters $a \in \Sigma$. For instance, typing heuristics for HTML forbid shared generalizations for different elements, which may be tables, rows, or lines, or the same elements with different attribute values.

**Interactive Learning.** All successful Web information extraction systems learn in an incremental manner [19,16,4]. This is essential for solving extraction tasks with a reasonably small number of user annotations. The algorithm $\mathtt{RPNI}^{S,\mathsf{type}}_{\mathsf{prune,cons}}$ can be used as core learning algorithm in an interactive environment such as Squirrel [4], in which it is repeatedly applied during user interaction.

Incremental learning algorithms help users to annotate a collection of Web pages. They always know a current hypothesis for the target query, which should solve the information extraction task at the end. At the beginning, this query can be chosen to be empty, i.e., $Q(t) = \emptyset$ for all $t \in T_\Sigma$. For every Web page, the user loops as follows in order to find complete annotations for all pages:

– apply the current query hypothesis to the current page,
– either accept the result and continue with the next page,
– or else, correct some of the errors by adding new partial annotations for the current page, learn a new query hypothesis by running algorithm $\mathtt{RPNI}^{\mathcal{S},\mathsf{type}}_{\mathsf{prune,cons}}$ with all complete annotations for earlier pages and all partial annotations for the current page, and repeat the procedure for the current page.

The quality of such interactive learning algorithms is usually measured in the number of pages that are to be annotated before the target query is found, and in the total number of corrections effected on these pages.

## 5 Experimental Results

We have implemented the learning algorithm $\mathtt{RPNI}^{\mathcal{S},\mathsf{type}}_{\mathsf{prune,cons}}$, and integrated it into the interactive environment Squirrel. Here we describe aspects of the implementation and results of experiments of schema-guided query induction for Web information extraction. The schema definition we use is the DTD of HTML.

**Preprocessing the DTD of HTML.** The complete DTD of HTML is huge. We have kept only the essential part with respect to information extraction. Indeed, HTML (XHTML1-transitional) DTD has 89 defined symbols. In practice, depending on the considered set of Web pages, the number of elements actually used ranges between 20 and 30. Reducing the size of the DTD can be done by filtering those elements and putting away the others and the unused rules of the schema automaton, i.e., rules that contain states that are not accessible or co-accessible. The automaton for the whole HTML DTD obtained by classical Glushkov's construction has 3951 rules. This reduction technique allows us to deal with an automaton whose number of rules ranges from 107 to 218 depending on the benchmark, and thus to speed up the inclusion tests.

**Implementing the Learner.** We have implemented $\mathtt{RPNI}^{\mathcal{S},\mathsf{type}}_{\mathsf{prune,cons}}$ in Objective CAML. Besides the usual efforts for implementing RPNI, a large part of the effort was spend on the inclusion test, which is done in an incremental manner. All parameters of the algorithm are provided, and can be freely instantiated (schema-guided pruning, schema-guided consistency, state typing heuristics). This allows us to measure the impact of these heuristics together or independently. No further heuristic has been introduced. This is quite important. It excludes all kinds of dirty tricks, so that the results can be obtained from the description presented here. As a drawback, it leaves some room for improving the performance.

**Benchmarks.** We have performed preliminary experiments on three benchmarks: Google, Okra and Bigbook[1]. Google presents a set of 34 result Web pages for the well-known search engine where links are to be extracted. Okra (251 pages) and Bigbook (234 pages) are classical benchmarks for data extraction

[1] Those benchmarks can be found at
`http://www.grappa.univ-lille3.fr/~carme/WebWiki/DataSets.html`

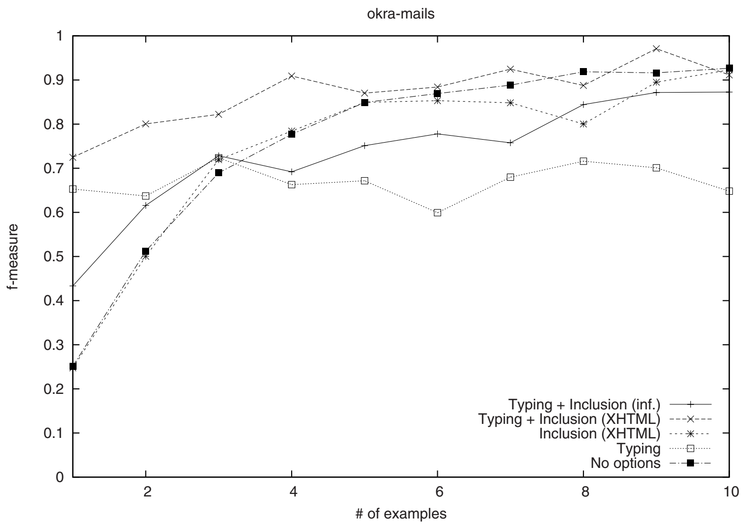


**Fig. 5.** Experimental results for Okra benchmark in non-interactive mode

on the Web. They both correspond to lists of people with several information on them. The task on Okra is to extract emails of persons, and names for Bigbook. We present here only Okra and Google because of space constraints. While a far simpler task, Bigbook results are comparable with those of Google in the way that the order of the different curves are similar, but with better overall results for every option set.

**Non-interactive Learning.** For the chosen benchmarks, a sample of 1 to 10 randomly chosen completely annotated Web pages is submitted to the learning algorithm. The resulting queries is tested on 30 other Web pages of the corpus using precision, recall and f-measure. The presented results are averages on 30 experiments.

Results are presented for different sets of options in Fig. 5 and 6. Results for pruning are not presented here because this option does not alter a lot the results in non-interactive learning. For inclusion with inferred DTDs, only the best result is presented (the one obtained when joint with typing).

From this, several conclusions can be raised. First, surprisingly, inclusion with inferred DTD leads to poor results. This might be related to the chosen DTD inference algorithm itself. Second, inclusion alone is not helpful, but joint with typing, it may be of serious help. In Okra benchmark, the learning algorithm with typing and inclusion within HTML DTD gives the best results, especially with few examples. On the other hand, results on Google (and Bigbook) do not really improve existing results.

Also, experiments on running time and number of merges performed have been done. Results vary but algorithm with schema consistency checking is usually around five time slower, which is still acceptable for an interactive use considering

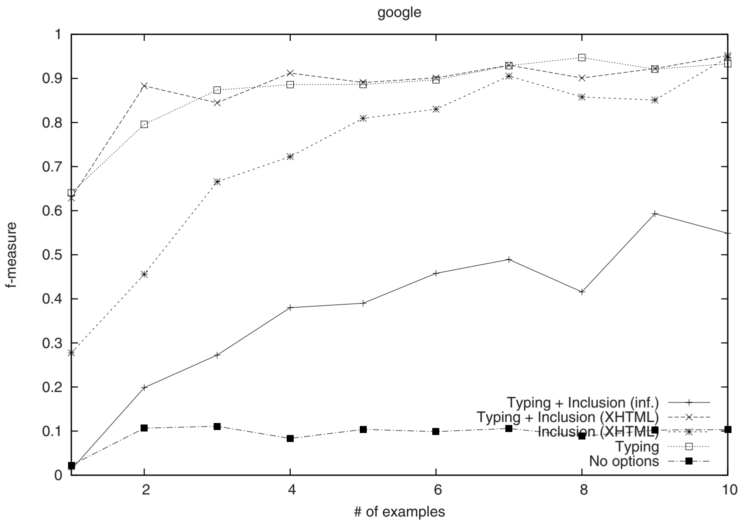


**Fig. 6.** Experimental results for Google benchmark in non-interactive mode

that the overall learning time rarely exceed one second. This proves the feasibility of the approach. On Okra, the schema consistency option allows to reduce the number of merges. This means that in this case, extra computation time can actually be compensated by a better guiding of merge operations. However, on other benchmarks, the number of merges is quite similar with or without schema consistency.

**Interactive Learning.** We now evaluate our algorithm in an interactive setting. It is tested automatically by a "user simulator" which performs the following task. We first begin with the empty query and a randomly chosen Web page. The query is tested on the Web page. If the result of the query is not correct, the user gives exactly one extra annotation (a correction): either a positive annotation on a node that the query forgot or a negative annotation on a node that is incorrectly annotated by the query. The annotation is chosen as being on the first node (in the document order) returned by the query that is incorrect. If the annotation returned by the query is correct, the user choses an other page and reiterate until it is satisfied, which is obtained when 30 consecutive Web pages are correctly annotated by this process.

During this protocol, we count the number of corrections the user had to do and the number of pages on which there has been an interaction (Web pages on which the query was already correct is not counted here). Also, once an annotated Web page has been accepted, we complete its annotation, i.e., we annotate negatively every non-annotated node. The results presented in Table 1 are averages on 30 experiments.

All the experiments have been performed using pruning.We present here the results obtained when adding typing, inclusion within HTML DTD, or both.

**Table 1.** Interactive learning. For each dataset, we present the number of necessary corrections/pages to learn the target query (T=typing heuristics; I=inclusion; P=schema-guided pruning). All experiments have been done with regular pruning, unless P is specified.

| | T | I (HTML DTD) | T + I (HTML DTD) | T + I (Inferred DTD) | T + P (HTML DTD) |
|---|---|---|---|---|---|
| Okra | failed | 17.93/3.87 | 4.00/2.03 | 4.60/2.73 | 3.73/1.87 |
| Bigbook | 3.03/1.37 | 3.20/1.57 | 2.77/1.77 | 2.33/1.33 | 3.90/1.37 |
| Google | 4.53/2.33 | 9.60/3.43 | 8.00/4.00 | 28.60/12.03 | 6.90/3.53 |

Also, we present results with typing and inclusion within the inferred DTD. Last, we tried inference using typing and pruning using schema, without using inclusion. Indeed, while inclusion is not possible with regular pruning (i.e., without schema), it is possible to prune trees by replacing them by their DTD state, without using inclusion. This is useful to separate the effect of this kind of pruning and the one of inclusion checking.

Those results are to be compared with other existing systems. The Squirrel system [4] learns correctly the query for Okra with 1.6 pages and 3.5 (average) corrections, 1 page and 3 corrections for Bigbook and 1.9 pages and 4.8 corrections for Google. Squirrel is basically the same algorithm as the one presented here, with options typing and pruning, but with several other heuristics and various optimizations. In [19], the (k,l)-contextual learning algorithm can infer the query for Okra and Bigbook with respectively 2 and 2.3 corrections (number of pages is not specified).

In the interactive setting, we can observe that there is a benchmark where inclusion really helps. On Okra, we have not been able to obtain decent results without this option. On the other hand, on Google and Bigbook, inclusion either does not give important improvement or even may lower performance a bit. Surprisingly, results of pruning with schema without using inclusion are a bit better (although maybe not significantly) than with inclusion. This could tend to indicate that, at least on observed benchmarks, the main use of the schema is actually in the pruning part rather than in the inclusion test.

**Acknowledgments.** We are grateful to Grégoire Laurence for his help on experiments. The work was supported by the Mostrare team-project of the INRIA Lille-Nord Europe research center, the Laboratoire d'Informatique Fondamentale de Lille (LIFL) UMR 8022 CNRS.

## References

1. Bex, G.J., Neven, F., Schwentick, T., Tuyls, K.: Inference of Concise DTDs from XML data. In: VLDB, pp. 115–126 (2006)
2. Brüggemann-Klein, A.: Regular Expressions to Finite Automata. Theoretical Computer Science 120(2), 197–213 (1993)

3. Brüggemann-Klein, A., Wood, D.: One-unambiguous Regular Languages. Information and Computation 142(2), 182–206 (1998)
4. Carme, J., Gilleron, R., Lemay, A., Niehren, J.: Interactive Learning of Node Selecting Tree Transducers. Machine Learning 66(1), 33–67 (2007)
5. Carme, J., Niehren, J., Tommasi, M.: Querying Unranked Trees with Stepwise Tree Automata. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 105–118. Springer, Heidelberg (2004)
6. Champavère, J., Gilleron, R., Lemay, A., Niehren, J.: Efficient Inclusion Checking for Deterministic Tree Automata and DTDs. In: LATA (to appear, 2008)
7. Cohen, W.W., Hurst, M., Jensen, L.S.: A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. In: WWW, pp. 232–241 (2002)
8. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (revised) (October 2007), http://www.grappa.univ-lille3.fr/tata
9. Coste, F., Fredouille, D., Kermovant, C., de la Higuera, C.: Introducing Domain and Typing Bias in Automata Inference. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 115–126. Springer, Heidelberg (2004)
10. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys 33(3), 374–425 (2001)
11. Finn, A., Kushmerick, N.: Multi-level Boundary Classification for Information Extraction. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 111–122. Springer, Heidelberg (2004)
12. Gilleron, R., Marty, P., Tommasi, M., Torre, F.: Interactive Tuples Extraction from Semi-structured Data. In: WI, pp. 997–1004 (2006)
13. Kosala, R.: Information Extraction by Tree Automata Inference. PhD thesis, K. U. Leuven (July 2003)
14. Kristjansson, T.T., Culotta, A., Viola, P., McCallum, A.: Interactive Information Extraction with Constrained Conditional Random Fields. In: AAAI (2004)
15. Lemay, A., Niehren, J., Gilleron, R.: Learning n-ary Node Selecting Tree Transducers from Completely Annotated Examples. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 253–267. Springer, Heidelberg (2006)
16. Lerman, K., Minton, S., Knoblock, C.: Wrapper Maintenance: a Machine Learning Approach. Journal of Artificial Intelligence Research 18, 149–181 (2003)
17. Oncina, J., Garcia, P.: Inferring Regular Languages in Polynomial Update Time. In: Pattern Recognition and Image Analysis, pp. 49–61 (1992)
18. Raeymaekers, S.: Information Extraction from Web Pages Based on Tree Automata Induction. PhD thesis, K. U. Leuven (January 2008)
19. Raeymaekers, S., Bruynooghe, M., Van den Bussche, J.: Learning (k,l)-contextual Tree Languages for Information Extraction. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 305–316. Springer, Heidelberg (2005)

# A Polynomial Algorithm for the Inference of Context Free Languages

Alexander Clark[1], Rémi Eyraud[2], and Amaury Habrard[2]

[1] Department of Computer Science,
Royal Holloway, University of London
alexc@cs.rhul.ac.uk
[2] Laboratoire d'Informatique Fondamentale,
University of Aix-Marseille, CNRS
{remi.eyraud,amaury.habrard}@lif.univ-mrs.fr

**Abstract.** We present a polynomial algorithm for the inductive inference of a large class of context free languages, that includes all regular languages. The algorithm uses a representation which we call Binary Feature Grammars based on a set of features, capable of representing richly structured context free languages as well as some context sensitive languages. More precisely, we focus on a particular case of this representation where the features correspond to contexts appearing in the language. Using the paradigm of positive data and a membership oracle, we can establish that all context free languages that satisfy two constraints on the context distributions can be identified in the limit by this approach. The polynomial time algorithm we propose is based on a generalisation of distributional learning and uses the lattice of context occurrences. The formalism and the algorithm seem well suited to natural language and in particular to the modelling of first language acquisition.

## 1 Introduction

For dealing with natural languages, there is a tension between using highly expressive formalisms and using formalisms that can be learned. For example, Tree Adjoining Grammars or other mildly context sensitive formalisms are very powerful but are difficult to handle from a machine learning standpoint. Many learnability results have been obtained for regular languages or for small subclasses of context free languages [1,2,3], but these results are still much too limited from a language theoretic point of view. In this paper, we propose to bridge for the first time the gap between theoretically well founded grammatical inference methods and the sorts of representations required for modelling natural languages. We present a family of representations for highly structured context free languages and show how they can be learned using a generalisation of distributional learning.

The contributions of this paper are as follows: We present in Section 3 a rich grammatical formalism, which we call *Binary Feature Grammars* (BFG). The class of languages defined by BFGs contains all context free languages (CFL) and some non context free languages. This makes the formalism a good candidate for representing natural languages. We will then show how the features can be defined in terms of the contexts of strings (Section 4), and how these grammars using context features can be learned directly from samples. We then prove in Section 5 our main result: that there is an algorithm that can efficiently identify in the limit the class of CFLs with certain properties, the finite context property and the finite kernel property, from positive data and a membership oracle.

## 2 Basic Definitions

We consider a finite alphabet $\Sigma$, and $\Sigma^*$ the free monoid generated by $\Sigma$. $\lambda$ is the empty string, and a language is a subset of $\Sigma^*$. We will write the concatenation of $u$ and $v$ as $uv$, and similarly for sets of strings. $u \in \Sigma^*$ is a substring of $v \in \Sigma^*$ if there are strings $l, r \in \Sigma^*$ such that $v = lur$. Define $Sub(u)$ to be the set of nonempty substrings of $u$. For a set of strings $S$ define $Sub(S) = \bigcup_{u \in S} Sub(u)$.

A context is an element of $\Sigma^* \times \Sigma^*$. For a string $u$ and a context $f = (l, r)$ we write $f \odot u = lur$; the insertion or wrapping operation. We extend this to sets of strings and contexts in the natural way. Define $Con(w) = \{(l, r) | \exists u \in \Sigma^+ : lur = w\}$; i.e. the set of all contexts of a word; similarly for a set of strings we define: $Con(S) = \bigcup_{w \in S} Con(w)$.

The set of contexts, or context distribution, of a string $u$ of a language $L$ is, $C_L(u) = \{(l, r) \in \Sigma^* \times \Sigma^* | lur \in L\}$. We will often drop the subscript where there is no ambiguity. We define the syntactic congruence as $u \equiv_L v$ iff $C_L(u) = C_L(v)$. The equivalence classes under this relation are the *congruence* classes of the language.

We now recall the definition of a context free grammar.

**Definition 1.** *A context free grammar (CFG) is a quadruple $G = (\Sigma, V, P, S)$. $\Sigma$ is a finite alphabet of terminal symbols, $V$ is a set of non terminals s.t. $\Sigma \cap V = \emptyset$, $P \subseteq V \times (V \cup \Sigma)^+$ is a finite set of productions, $S \in V$ is the start symbol.*

We denote a production of $P$: $N \rightarrow \alpha$ with $N \in V$ and $\alpha \in (V \cup \Sigma)^+$. We will write $uNv \Rightarrow_G u\alpha v$ if there is a production $N \rightarrow \alpha$ in $G$. $\overset{*}{\Rightarrow}_G$ denotes the reflexive transitive closure of $\Rightarrow_G$.

The language defined by a CFG $G$ is $L(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow}_G w\}$. In general we will assume that $\lambda$ is not a member of any language.

## 3 Binary Feature Grammars

Before the presentation of our formalism, we give some results about contexts that will help give an intuition about the representation. A standard lemma is:

**Lemma 1.** *For any language $L$ and for any strings $u, u', v, v'$ if $C(u) = C(u')$ and $C(v) = C(v')$, then $C(uv) = C(u'v')$.*

This establishes that the syntactic monoid $\Sigma^* / \equiv_L$ is well-defined; from a learnability point of view this means that if we want to compute the contexts of a string $w$ we can look for a split into two strings $uv$ where $u$ is congruent to $u'$ and $v$ is congruent to $v'$; if we can do this and we know how $u'$ and $v'$ combine, then we know that the contexts of $uv$ will be exactly the contexts of $u'v'$. There is also a slightly stronger result:

**Lemma 2.** *For any language $L$ and for any strings $u, u', v, v'$ if $C(u) \subseteq C(u')$ and $C(v) \subseteq C(v')$, then $C(uv) \subseteq C(u'v')$.*

*Proof.* We write out the proof completely as the ideas will be used later on: suppose we have $u, v, u', v'$ that satisfy the conditions. Suppose $(l, r) \in C(uv)$; then $(l, vr) \in C(u)$ and therefore $(l, vr) \in C(u')$. So $(lu', r) \in C(v)$, and therefore $(lu', r) \in C(v')$, so $(l, r) \in C(u'v')$. □

Looking at Lemma 2 we can also say that, if we have some finite set of strings $K$, where we know the contexts, then:

**Corollary 1**

$$C(w) \supseteq \bigcup_{\substack{u',v':\\ u'v'=w}} \bigcup_{\substack{u\in K:\\ C(u)\subseteq C(u')}} \bigcup_{\substack{v\in K:\\ C(v)\subseteq C(v')}} C(uv)$$

This is the basis of our representation: a word $w$ is characterised by its set of contexts. We can compute the representation of $w$, from the representation of its parts $u', v'$, by looking at all of the other matching strings $u$ and $v$ where we understand how they combine (with subset inclusion). Rather than representing just the congruence classes, we will represent the lattice structure of the set of contexts using subset inclusion; sometimes called Dobrušin-domination [4].

To express this basic idea of inference, we will define an appropriate formalism: *binary feature grammars*. Initially, we will define it with no reference to learnability. Note the resemblance between this formalism and GPSG [5], and most importantly the class of Range Concatenation Grammars (RCG) [6].

**Definition 2.** *We define a Binary Feature Grammar (BFG) $G$ as a tuple $\langle F, f_s, P, P_L, \Sigma\rangle$. $F$ is a finite set (of features), where we write $C = 2^F$ for the power set of $F$ defining the categories of the grammar, $P \subseteq C \times C \times C$ is a finite set of productions that we write $x \to yz$ where $x, y, z \in C$ and $P_L \subseteq C \times \Sigma$ is a set of lexical rules, written $x \to a$ and $f_s \in F$ is the sentence feature.*

Normally $P_L$ will contain exactly one production for each letter in the alphabet.

A BFG $G$ defines recursively a map $f_G$ from $\Sigma^* \to C$ as follows:

$$f_G(\lambda) = \emptyset \tag{1}$$

$$f_G(w) = \bigcup_{(c\to w)\in P_L} c \qquad \text{iff } |w| = 1 \tag{2}$$

$$f_G(w) = \bigcup_{u,v:uv=w} \bigcup_{\substack{x\to yz\in P:\\ y\subseteq f_G(u)\wedge\\ z\subseteq f_G(v)}} x \qquad \text{iff } |w| > 1. \tag{3}$$

Note the relation between the third clause above and Corollary [1]. Note also that in general we will apply more than one production at each step of the analysis. Given a BFG $G$ and a string $w$ it is possible to compute $f_G(w)$ in time $\mathcal{O}(|F||P||w|^3)$ using standard dynamic programming techniques.

**Definition 3.** *The language defined by a BFG $G$ is the set of all strings that are assigned the sentence feature: $L(G) = \{u | f_s \in f_G(u)\}$.*

While this formalism has some relationship to a context free grammar, and some to a semi-Thue system (also known as a string rewriting system), it is not formally identical to either of these. The only exact equivalence is to a restricted subset of Range Concatenation Grammars; a very powerful formalism [6]. We include the following relationship, but suggest that the reader unfamiliar with RCGs proceeds to the discussion of the relationship with the more familiar class of context free grammars.

**Lemma 3.** *For every BFG G, there is a non-erasing positive range concatenation grammar of arity one, in 2-var form that defines the same language.*

*Proof.* Suppose $G = \langle F, f_s, P, P_L \rangle$. Define a RCG with a set of predicates equal to $F$ and the following clauses, and the two variables $U, V$. For each production $x \to yz$ in $P$, for each $f \in x$, where $y = \{g_1, \dots g_i\}$, $z = \{h_1, \dots h_j\}$ add clauses

$$f(UV) \to g_1(U), \dots g_i(U), h_1(V), \dots h_j(V).$$

For each lexical production $\{f_1 \dots f_k\} \to a$ add clauses

$$f_i(a) \to \epsilon.$$

It is straightforward to verify that $f(w) \vdash \epsilon$ iff $f \in f_G(w)$. □

## 3.1 BFGs and CFGs

Let $G = \langle V, S, P, \Sigma \rangle$ be a CFG in Chomsky Normal Form (CNF). We will now construct an equivalent BFG grammar $G_{BFG} = \langle F, f_s, \hat{P}, P_L, \Sigma \rangle$ where $F = V$, $f_s = S$ and $\hat{P} = \{\{X\} \to \{Y\}\{Z\} | X \to YZ \in P\}$, and $P_L = \{\{X\} \to a | X \to a \in P\}$. We claim that the BFG $G_{BFG}$ defines the same language as $G$.

**Lemma 4.** *Let $G = \langle V, S, P, \Sigma \rangle$ be a CFG in CNF. For every string $w$, $N \overset{*}{\Rightarrow} w$ if and only if $N \in f_{G_{BFG}}(w)$.*

Note that for this construction, computing the feature map $f_G$ is exactly equivalent to computing the CKY parse table.

**Lemma 5.** *Every CF language can be represented by a BFG.*

*Proof.* Let $G$ a CFG and $G_{BFG}$ the BFG described above. By Lemma [4], if $S \Rightarrow^* w$ then $S = f_S \in f_{G_{BFG}}(w)$ and vice versa; thus $L(G) = L(G_{BFG})$. □

**BFG and Non context free languages.** BFGs are more powerful than CFGs in two respects. First, BFGs can compactly represent languages like the finite language of all $n!$ permutations of an $n$-letter alphabet, that have no concise representation as a CFG [7]. Secondly, BFGs can represent some non-context free languages. Let $L = \{a^n b^n c^n d | n > 0\}$, which is clearly not context free. However we can construct a BFG that recognises this language. Let $G = \langle F, S, P, P_L, \Sigma\rangle$ a BFG s.t.:

- $F = \{A, A', B, C, C', AB, AB', AAB, BC, BC', BBC, D, S\}$,
- $P = \{\{S\} \to \{AB', BC'\}\{D\}, \{AB'\} \to \{AB\}\{C'\},$
  $\{AB\} \to \{AAB\}\{B\}, \{AB\} \to \{A\}\{B\},$
  $\{AAB\} \to \{A\}\{AB\}, \{BC'\} \to \{A'\}\{BC\},$
  $\{BC\} \to \{BBC\}\{C\}, \{BC\} \to \{B\}\{C\}, \{BBC\} \to \{B\}\{BC\},$
  $\{A'\} \to \{A'\}\{A\}, \{C'\} \to \{C'\}\{C\}\}$,
- $P_L = \{\{A, A'\} \to a, \{B\} \to b, \{C, C'\} \to c, \{D\} \to d\}$.

To give an intuition on the construction of the grammar, we describe the contribution of some features:

- $AB$ defines the language $\{a^n b^n | n > 0\}$
- $AB'$ defines the language $\{a^n b^n c^m | m > 0, n > 0\}$,
- $AAB$ defines the language $\{aa^n b^n | n > 0\}$,
- $BC'$ defines the language $\{a^m b^n c^n | m > 0, n > 0\}$,
- $BC$ defines the language $\{b^n c^n | n > 0\}$,
- $BBC$ defines the language $\{bb^n c^n | n > 0\}$.

## 3.2 Contextual Binary Feature Grammars

The class of BFGs is a powerful formalism: we are interested in a special case where the features are contexts. Here we define this in the most straightforward way though there are a number of obvious extensions.

**Definition 4.** *A Contextual Binary Feature Grammar is a BFG where the feature set is a finite set of contexts (i.e. $F \subset \Sigma^* \times \Sigma^*$) and the sentential feature is $(\lambda, \lambda)$.*

By itself this is not a constraint but we are interested in cases where there is a correspondence between the language theoretic interpretation of a context, and the occurrence of that context as a feature in the grammar: in this case the features will be observable which will lead to learnability. Clearly from an inference point of view, at the minimum we want the sentence features to be correct: if we are learning a target language $L$, then if $(\lambda, \lambda) \in f_G(u)$ iff $u \in L$ then $L(G) = L$ which is what we want. But ideally we also want $f_G$ to be correct for all features.

**Definition 5.** *Given a finite set of contexts $F = \{(l_1, r_1), \ldots, (l_n, r_n)\}$ and a language $L$ we can define the context feature map $F_L : \Sigma^* \to 2^F$ which is just the map $u \mapsto \{(l, r) \in F | lur \in L\} = C_L(u) \cap F$.*

Using this definition, we now need a correspondence between the language theoretic context feature map $F_L$ and the representation in our CBFG, $f_G$.

**Definition 6.** *A CBFG $G$ is* exact *if for all $u \in \Sigma^*$, $f_G(u) = F_{L(G)}(u)$.*

*Example.* Let $L = \{a^n b^n | n > 0\}$. Let $\langle F, (\lambda, \lambda), P, P_L, \Sigma \rangle$ a CBFG s.t. $F = \{(\lambda, \lambda), (a, \lambda), (aab, \lambda), (\lambda, b), (\lambda, abb)\}$. The lexical productions in $P_L$ are: $\{(\lambda, b), (\lambda, abb)\} \rightarrow a$ and $\{(a, \lambda), (aab, \lambda)\} \rightarrow b$. Then, the productions in $P$ are defined by the set: $\{(\lambda, \lambda)\} \rightarrow \{(\lambda, b)\}\{(aab, \lambda)\}$, $\{(\lambda, \lambda)\} \rightarrow \{(\lambda, abb)\}\{(a, \lambda)\}$, $\{(\lambda, b)\} \rightarrow \{(\lambda, abb)\}\{(\lambda, \lambda)\}$, $\{(a, \lambda)\} \rightarrow \{(\lambda, \lambda)\}\{(aab, \lambda)\}$.

This defines an exact CBFG for $L$.

Clearly every exact CBFG is a BFG, but we conjecture that the class of languages with an exact CBFG is strictly smaller than the class of languages defined by general BFGs.

## 4 Inference

We have carefully defined the representation so that the inference algorithm will be almost trivial. Given a set of strings, and a set of contexts, we can simply write down a CBFG that will approximate a particular language.

**Definition 7.** *Let $F$ be the set of contexts, $(\lambda, \lambda) \in F$, $K$ a finite set of strings, $P_L = \{F_L(u) \rightarrow u | u \in K \wedge |u| = 1\}$ and $P = \{F_L(uv) \rightarrow F_L(u)F_L(v) | u, v, uv \in K\}$. We define $G_0(K, L, F)$ as the CBFG $\langle F, (\lambda, \lambda), P, P_L, \Sigma \rangle$.*

We will call $K$ here the basis for the language. The set of productions is defined merely by observation: we take the set of all productions that we observe as the concatenation of elements of the small set $K$. Often $K$ will be closed under substrings: i.e. $Sub(K) = K$. This grammar is a CBFG but in general it will not be exact. For example, the language it defines might be empty, in which case $F_L(u) = \emptyset$ for all $u$, and yet it could define some features on the grammar.

Clearly the language defined depends on two factors: the set of strings $K$ and the set of features $F$. We now establish two important lemmas: first, that as $K$ increases the language defined by $G_0(K, L, F)$ will increase, and secondly that as $F$ increases the language will decrease.

**Lemma 6.** *Suppose we have two CBFGs defined by $G = G_0(K, L, F)$ and $G' = G_0(K, L, F')$ where $F \subseteq F'$. Then for all $u$, $f_G(u) \supseteq f_{G'}(u) \cap F$.*

*Proof.* Let $G'$ have a set of productions $P', P'_L$, and $G$ have a set of productions $P, P_L$. Clearly if $x \rightarrow yz \in P'$ then $x \cap F \rightarrow (y \cap F)(z \cap F)$ is in $P$ by the definition of $G_0$, and likewise for $P_L, P'_L$. By induction on $|u|$ we can show that any feature in $f_{G'}(u) \cap F$ will be in $f_G(u)$. The base case is trivial since $F'_L(a) \cap F = F_L(a)$; if it is true for all strings up to length $k$, then if $f \in f_{G'}(u) \cap F$; there must be a production in $F'$ with $f$ on the head. By the inductive hypothesis, the right hand sides of the corresponding production in $P$ will be triggered, and so $f$ must be in $f_G(u)$. □

**Corollary 2.** *Suppose we have two CBFGs defined by $G = G_0(K, L, F)$ and $G' = G_0(K, L, F')$ where $F \subseteq F'$; then $L(G) \supseteq L(G')$.*

Conversely, we can show that as we increase $K$, the language and the map $f_G$ will increase. This is addressed by the next lemma.

**Lemma 7.** *Suppose we have two CBFGs defined by $G = G_0(K, L, F)$ and $G' = G_0(K', L, F)$ where $K \subseteq K'$. Then for all $u$, $f_{G_0(K,L,F)}(u) \subseteq f_{G_0(K',L,F)}(u)$.*

*Proof.* Clearly the sets of productions of $G_0(K, L, F)$ will be a subset of the set of productions of $G_0(K', L, F)$, and so anything that can be derived by the first can be derived by the second, again by induction on the length of the string. □

To establish learnability, we need to prove that for a target language $L$, if we have a sufficiently large $F$ then $L(G_0(K, L, F))$ will be contained within $L$ and that if we have a sufficiently large $K$, then $L(G_0(K, L, F))$ will contain $L$.

## 4.1 Fiducial Feature Sets and Finite Context Property

We need to be able to prove that for any $K$ if we have enough features then the language defined will be included within the target language $L$. We formalise the idea of having enough features in the following way:

**Definition 8.** *For a language $L$ and a string $u$, a set of features $F$ is fiducial on $u$ if for all $v \in \Sigma^*$, $F_L(u) \subseteq F_L(v)$ implies $C_L(u) \subseteq C_L(v)$.*

Note that if $F$ is fiducial on $u$ and $F \subset F'$ then $F'$ is fiducial on $u$. Therefore we can naturally extend this to sets of strings.

**Definition 9.** *For a set of strings $K$, a set of features $F$ is fiducial if for all $u \in K$, $F$ is fiducial on $u$.*

Note the asymmetry between $u$ and $v$ in these definitions. If $u$ and $v$ are both in $K$ then having the same features means they are syntactically congruent. However if two strings, neither of which are in $K$, have the same features this does not mean they are necessarily congruent (for instance if $F_L(v) = F_L(v') = \emptyset$). For non finite state languages, the set of congruence classes will be infinite, and thus we cannot have a finite fiducial set for the set of all strings in $Sub(L)$, but we can have a feature set that is correct for a finite subset of strings, or more generally for an infinite set of strings, if they fall into a finite number of congruence classes.

We now define the finite context property.

**Definition 10.** *A language $L$ has the Finite Context Property (FCP) if every string has a finite fiducial feature set.*

Clearly if $L$ has the FCP, then any finite set of substrings, $K$, has a finite fiducial feature set which will be the union of the finite fiducial feature sets for each element of $K$. If $u \notin Sub(L)$ then any set of features is fiducial since $C_L(u) = \emptyset$.

Not all CFL have the FCP: for instance $L = \{a^n b | n > 0\} \cup \{a^n c^m | n > m > 0\}$, does not have the FCP, since there is no finite fiducial feature set for the string $b$; for any such set there will be some $N$ such that $c^N$ will have all of those features, but $C_L(c^N)$ is not a superset of $C_L(b)$.

However, all regular languages have the FCP since they have a finite number of syntactic congruence classes.

We can now state the most important lemma: this lemma links up the definition of the feature map in a BFG, with the fiducial set of features to show that only correct features will be assigned to substrings by the grammar. It states that the features assigned by the grammar will correspond to the language theoretic interpretation of them as contexts.

**Lemma 8.** *For any language $L$, given a set of strings $K$ and a set of features $F$, let $G = G_0(K, L, F)$. If $F$ is fiducial on $K$, then for all $w \in \Sigma^*$ $f_G(w) \subseteq F_L(w)$.*

*Proof.* We proceed by induction on length of the string. *Base case:* strings of length 1. $f_G(w)$ will be the set of observed contexts of $w$, and since we have observed these contexts, they must be in the language. *Inductive step:* let $w$ a string of length $k$. Take a feature $f$ on $f_G(w)$; by definition this must come from some production $x \rightarrow yz$ and a split $u, v$ of $w$. The production must be from some elements of $K$, $u', v'$ and $u'v'$ such that $y = F_L(u'), z = F_L(v')$ and $x = F_L(u'v')$. If the production applies this means that $F_L(u') = y \subseteq f_G(u) \subseteq F_L(u)$ (by inductive hypothesis), and similarly $F_L(v') \subseteq F_L(v)$. By fiduciality of $F$ this means that $C(u') \subseteq C(u)$ and $C(v') \subseteq C(v)$. So by Lemma 2 $C(u'v') \subseteq C(uv)$. Since $f \in C(u'v')$ then $f \in C(uv) = C(w)$. Therefore, since $f \in F$ and $C(w) \cap F = F_L(w)$, $f \in F_L(w)$, and therefore $f_G(w) \subseteq F_L(w)$. □

**Corollary 3.** *If $F$ is fiducial on $K$, and $(\lambda, \lambda) \in F$ then $L(G_0(K, F, L)) \subseteq L$.*

Therefore for any finite set $K$ from an FCP language, we can find a set of features so that the language defined by those features on $K$ is not too big.

## 4.2 Kernel and Finite Kernel Property

We will now show a complementary result, namely that for a sufficiently large $K$ the language defined by $G_0$ will include the target language.

**Definition 11.** *A finite set $K \subseteq \Sigma^*$ is a kernel for a language $L$, if for any set of features $F$, $L(G_0(K, F, L)) \supseteq L$.*

To prove that a set is a kernel, it suffices to show that a fiducial set of features will define the language; any smaller set of features define then a larger language. In fact we can take the infinite set of all contexts and define productions based on the congruence classes. If $F$ is the set of all contexts then we have $F_L(u) = C_L(u)$, thus the productions will be exactly of the form $C(uv) \rightarrow C(u)C(v)$.

This is a slight abuse of notation since feature sets are normally finite.

**Lemma 9.** *Let $F = \Sigma^* \times \Sigma^*$; if $L(G_0(K, L, F)) \supseteq L$ then $K$ is a kernel.*

*Proof.* By monotonicity of $F$: any finite feature set will be a subset of $F$. □

Not all context free languages will have a finite kernel. For example $L = \{a^+\} \cup \{a^n b^m | n < m\}$ does not have a finite kernel, but is clearly CF. Indeed, assume that the a set $K$ contains all strings of length less than or equal to $k$. Assume w.l.o.g. that the fiducial set of features for $K$ includes all features $(\lambda, b^i)$, where $i \leq k+1$. Consider the rules of the form $F_L(a^k) \rightarrow F_L(a^j)F_L(a^{k-j})$; it is easy to see that no matter how large $k$ is, the derived CBFG will undergenerate as $a^k$ is not congruent to $a^{k-1}$.

**Definition 12.** *A context free grammar $G_T = \langle V, S, P, \Sigma \rangle$ has the Finite Kernel Property (FKP) iff for every non-terminal $N \in V$ there is a finite set of strings $K(N)$ such that for all $k \in K(N), N \overset{*}{\Rightarrow} k$ and where for every string $w \in \Sigma^*$ such that $N \overset{*}{\Rightarrow} w$ there is a string $k \in K(N)$ such that $C(k) \subseteq C(w)$. A CFL $L$ has the FKP, if there is a grammar in CNF for it with the FKP. We also assume that $a \in K(N)$ if $a \in \Sigma$ and $N \rightarrow a \in P$.*

Notice that all regular languages have the FKP since they have a finite number of congruence classes.

**Lemma 10.** *Any context free language with the FKP has a finite kernel.*

*Proof.* Let $G_T = \langle V, S, P, \Sigma \rangle$ be such a CNF CFG with the FKP. Define

$$K(G_T) = \bigcup_{N \in V} \left( K(N) \cup \bigcup_{X \rightarrow MN \in P} K(M)K(N) \right). \qquad (4)$$

We claim that $K(G_T)$ is a kernel. Assume that $F = \Sigma^* \times \Sigma^*$. Let $G = G_0(K(G_T), L(G_T), F) = \langle F, (\lambda, \lambda), P, P_L, \Sigma \rangle$.

We will show, by induction on the length of derivation of $w$ in $G_T$, that for all $N, w$ if $N \overset{*}{\Rightarrow} w$ then there is a $k$ in $K(N)$ such that $f_G(w) \supseteq C(k)$. If length of derivation is 1, then this is true since $|w| = 1$ and thus $w \in K(N)$: therefore $C(w) \rightarrow w \in P_L$. Suppose it is true for all derivations of length less than $j$. Take a derivation of length $j$; say $N \overset{*}{\Rightarrow} w$. There must be a production in $G_T$ of the form $N \rightarrow PQ$, where $P \Rightarrow^* u$ and $Q \Rightarrow^* v$, and $w = uv$. By inductive hypothesis; we have $f_G(u) \supseteq C(k_u)$ and $f_G(v) \supseteq C(k_v)$. By construction $k_u k_v \in K(G_T)$ and then there will be a rule $C(k_u k_v) \rightarrow C(k_u)C(k_v)$ in $P$. Therefore $f_G(uv) \supseteq C(k_u k_v)$. Since $N \overset{*}{\Rightarrow} k_u k_v$ there must be some $k_{uv} \in K(N)$ such that $C(k_{uv}) \subseteq C(k_u k_v)$. Therefore $f_G(w) \supseteq C(k_u k_v) \supseteq C(k_{uv})$. □

Now we can see that if $w \in L$, then $S \overset{*}{\Rightarrow} w$, then there is a $k \in K(S)$ such that $f_G(w) \supseteq C(k)$ and $S \overset{*}{\Rightarrow} k$, therefore $(\lambda, \lambda) \in f_G(w)$ since $(\lambda, \lambda) \in C(k)$, thus $w \in L(G)$ and therefore $K$ is a kernel.

## 5 Algorithm

Before we present the algorithm, we will discuss the learning model that we use. The class of languages that we will learn is suprafinite and thus we cannot get a straight identification in the limit (IIL) result [8]. Ultimately we are interested in a more realistic probabilistic learning paradigm, but for mathematical convenience it is appropriate to establish the basic results in a symbolic paradigm. The ultimate goal is to model natural languages, where negative data, or equivalence queries are generally not available or are computationally impossible. Accordingly, we have decided to use the model of positive data together with membership queries: an oracle can tell the learner whether a string is in the language or not [9]. The presented algorithm runs in time polynomial in the size of the sample $S$: since the strings are of variable length, this size must be the sum of the lengths of the strings in $S$, $\sum_{w \in S} |w|$.

We should note that this is not a strong enough result: [10] showed that any algorithm can be made polynomial, by only processing a small prefix of the data. It is hard to tighten the model sufficiently: the suggestion in [11] for a polynomial characteristic set is inapplicable for representations, such as the ones in this paper, that are powerful enough to define languages whose shortest strings are exponentially long. We note that the situation is unsatisfactory, but we do not intend to propose a solution in this paper. We merely point out that the algorithm is genuinely polynomial, processes all of the data in the sample without delaying tricks, is conservative and always produces a hypothesis that is compatible with the observed data and answers from the oracle.

Before we present the algorithm we hope that it is intuitively obvious how the approach will work. Figure 1 shows the relationship between $K$ and $F$. When we have a large enough $K$, we will be to the right of the vertical line; when we have enough features for that $K$ we will be above the diagonal line. Thus the basis of the algorithm is to move to the right, until we have enough data, and then to move up vertically, increasing the feature set until we have a fiducial set.

We can now define our learning algorithm in Algorithm 1. Informally, $D$ is the list of all strings that have been seen so far: the algorithm examines the

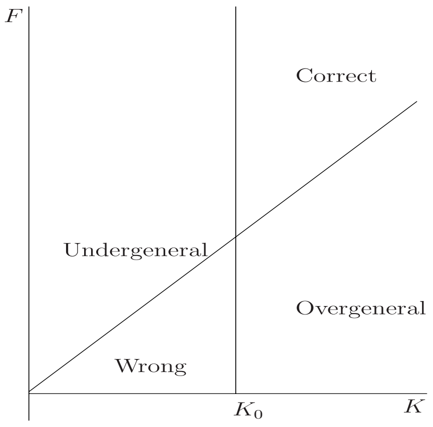


**Fig. 1.** The relationship between $K$ and $F$: The diagonal line is the line of fiduciality: above this line means that $F$ is fiducial on $K$. $K_0$ is the (a) kernel for the language.

**Algorithm 1.** BFG learning algorithm IIL

```
**Data**: A sequence of strings $S = \{w_1, w_2 \ldots, \}$, membership oracle $\mathcal{O}$
**Result**: A sequence of CBFGs $G_1, G_2, \ldots$
$K \leftarrow \emptyset$ ; $D \leftarrow \emptyset$ ; $F \leftarrow \{(\lambda, \lambda)\}$ ; $G_0 = G_0(K, \mathcal{O}, F)$ ;
for $w_i$ do
    $D \leftarrow D \cup \{w_i\}$; $T \leftarrow Con(D) \odot Sub(D)$ ;
    if $\exists w \in T$ such that $w \in L(G_{i-1}) \setminus L$ then $F \leftarrow Con(D)$ ;
    if $\exists w \in T$ such that $w \in L \setminus L(G_{i-1})$ then $K \leftarrow Sub(D)$ ; $F \leftarrow Con(D)$ ;
    Output $G_i = G_0(K, \mathcal{O}, F)$ ;
end
```

set of strings $T = Con(D) \odot Sub(D)$. If the current hypothesis generates some element of this set that is not in the language, then it is overgeneralising: we need to add features. If on the other hand we undergeneralise, then we add all of the substrings of $D$ to $K$, and all possible contexts to $F$. In Algorithm 1, $G_0(K, \mathcal{O}, F)$ denotes the same construction as $G_0(K, L, F)$, except that we use membership queries with the oracle $\mathcal{O}$ to compute $F_L$ for each element in $K$.

**Theorem 1.** *Algorithm 1 runs in polynomial time in the size of the sample, and makes a polynomial number of calls to the membership oracle.*

*Proof.* The value of $D$ will just be the set of observed strings; $Sub(D)$ and $Con(D)$ are both polynomially bounded by the size of the sample, so the number of calls to the oracle is clearly polynomial. Computing the feature map is polynomial in the length of the strings, and computing $G_0$ is also polynomial, since $K$ and $F$ will also be polynomially bounded. □

In the following, we consider the class of context free languages having the FCP and the FKP, represented by CBFG. $K_n$ denotes the value of $K$ at the $n^{th}$ loop, and similarly for $F$, $D$ and $T$, which is the test set; called by [12] the "explosion".

Clearly, if the grammar undergeneralises, when it encounters a string in $L(G) \setminus L$ it will increase the kernel. The corresponding fact for overgeneralisation is stated in this lemma:

**Lemma 11.** *For a given positive presentation of a CFL with the FCP and the FKP, if there is an $m$ such that $L \subset L(G_m)$ and $L \neq L(G_m)$, then there is a string $w \in L(G_m) \setminus L$ such that there exists an $n \geq m$ such that $w \in T_n$.*

*Proof.* Let $w$ be a shortest string such that there is a feature $f \in f_{G_m}(w) \setminus F_L(w)$. We know that this set is non empty as there is some string with $f = (\lambda, \lambda)$.

If $w \in Sub(L)$, then let $f'$ be some feature in $C_L(w)$, et $n$ be the smallest number such that $f' \odot w \in D_n$. $T_n$ must contain $f \odot w$, which satisfies the lemma.

Alternatively suppose $w \notin Sub(L)$, then the feature $f$ must come from some rule acting upon $uv = w$, where $u, v \in Sub(L)$ (since $w$ is a shortest string, all rules will have non empty features by construction of $G$). Let $(u'v') \rightarrow (u')(v')$ be one of the triples of strings in $K_m$ that produced this rule. Since $C(u'v') \not\subset C(uv)$

by Lemma 2 we have $C(u') \not\subset C(u)$, or $C(v') \not\subset C(v)$. Suppose w.l.o.g. that it is $u$, and consider a feature from $f'' \in C(u') \setminus C(u)$. Clearly $f'' \in Con(L)$, $u \in Sub(L)$ and $f'' \odot u$ is in $L(G_m) \setminus L$. Let $n$ be the smallest index where we have $u \in Sub(D_n)$ and $f'' \in Con(D_n)$; and then $f'' \odot u$ satisfies the lemma. □

**Lemma 12.** *For every positive presentation of a CFL $L$ with the FCP and the FKP, there is some $n$ such that either $L(G_n) = L$ or $K_n$ is a kernel, and $\forall N > n$ $K_N = K_n$.*

*Proof.* First of all if $K_n$ is a kernel, then $L \setminus L(G_n) = \emptyset$, and so the the algorithm will not add any more strings to the kernel. Let $m$ be the smallest number such that $Sub(D_m)$ is a kernel. Recall that any superset of a kernel is a kernel, and that all CFL with the FKP have a finite kernel (Lemma 10) so such an $m$ must exist. Consider the grammar $G_m$; there are three possibilities:

1. $L(G_m) = L$; in which case the grammar has converged.
2. $L \setminus L(G_m) \neq \emptyset$, in which case, let $w \in L \setminus L(G_m)$, and let $n$ be the first index of the presentation such that $w_n = w$; then $K_n$ must contain $Sub(D_m)$, since at some point the kernel must have been expanded, and therefore $K_n$ is a kernel.
3. $L(G_m) \setminus L(G) \neq \emptyset$, but $L \subset L(G_m)$. Let $F$ be a finite fiducial feature set for $K_m$, (we assume w.l.o.g. that all such $F$ are in $Con(K_m)$), and consider the smallest $m'$ such that $(F \odot K_m) \cap L \subset D_{m'}$. Consider $G_{m'}$, either it is correct or undergeneralises, in which case we apply the same argument we just used. Otherwise, it strictly overgeneralises, in which case by Lemma 11, we will expand the feature set to a set which by construction will be fiducial. At this point we will either have enlarged $K$, in which case it will be a kernel, or we have not, in which case we will either undergeneralise or be exact; if it undergeneralises, then when we observe some incorrect string, the basis will be enlarged to a kernel. □

**Lemma 13.** *For every positive presentation of a language $L$ with the FCP and the FKP, let $n$ be the smallest number such that $K_n$ is a kernel, if there is such an $n$. There is some $n_2 \geq n$ such that for all $N > n_2$, $F_N = F_{n_2}$ and the $L(G_0(K_N, L, F_N)) = L$.*

*Proof.* Let $F$ be a finite fiducial feature set for $K_n$. Let $n_1$ the smallest number s.t. $D_{n_1} \supseteq (F \odot K_n) \cap L$. Either $G_{n_1}$ is correct, in which case it is done, or it overgeneralises in which case by Lemma 11 there will be an $n_2$ which triggers the enlargement of the feature set. Since $F \subseteq Con(D_{n_1})$, the feature set will be fiducial. □

**Theorem 2.** *Algorithm 1 identifies in the limit the class of context free languages with the finite context property and the finite kernel property.*

*Proof.* Immediate by the preceding two lemmas. □

## 6 Discussion

First of all, we should establish how large the class of languages with the FCP and the FKP is: it includes all finite languages and all regular languages, since the set of congruence classes is finite for finite state languages. It similarly includes the context-free substitutable languages, [3], since every string in a substitutable language belongs to only one syntactic congruence class.

As already stated it does not include all CFLs since not all CFLs have the FCP and/or the FKP. However it does include languages like the Dyck languages of arbitrary order, Lukacevic language and most other classic simple examples.

As a special case consider the equivalence relation between contexts $f \cong_L f'$ iff $\forall u$ we have that $f \odot u \in L$ iff $f' \odot u \in L$. The class of CFLs where the context distribution of every string is a finite union of equivalence classes of contexts clearly has both the FKP and the FCP.

Our approach to context free grammatical inference is based on a generalisation of distributional learning, following the work of [3]. The current state of the art in context free inductive inference from flat examples only has been rather limited. When learning from stochastic data or using a membership oracle, it is possible to have powerful results, if we allow exponential computation (see for example [13]). The main contribution of this paper is to show that efficient learning is possible, with an appropriate representation. We currently rely on using a membership oracle, but under suitable assumptions about distributions, it should be possible to get a PAC-learning result for this class along the lines of [14], placing some bounds on the number of features required.

**Linguistics.** The field of grammatical inference has close relations to the study of language acquisition. Attempts to model natural languages with context free grammars require additional machinery: natural language categories such as noun phrases contain many overlapping subclasses with features such as case, number, gender and similarly for verbal categories. Modelling this requires either an exponential explosion of the number of non-terminals employed or a switch to a richer set of features. In our formalism, motivated by normal CF inference, we get this additional power for free. While we have implemented the algorithm described here, and verified that it works in accordance with theory on small artificial examples, there are a number of modifications that would need to be made before it can be applied to real grammar induction on natural language. First, the algorithm is very naive; in practice a more refined algorithm could select both the kernel and the feature set in a more sophisticated way. Secondly, considering features that correspond to individual contexts may be too narrow a definition for natural language given the well known problems of data sparseness and it will be necessary to switch to features corresponding to sets of contexts, which may overlap. Thus for example one might have features that correspond to sets of contexts of the form $F(u, v) = \{(lu, vr)|l, r \in \Sigma^*\}$. This would take this approach closer to methods that have been shown to be effective in unsupervised learning in NLP[15] where typically $|u| = |v| = 1$. In any event, we think such modifications will be necessary for the acquisition of non context free languages. Finally, at the moment the algorithm has polynomial update time, but in the worst case, there are deterministic finite state automata such that the size of the smallest kernel will be exponential in the number of states. There are, however, natural algorithms for generalising the productions by removing features from the right hand sides of the rules; this would have the effect of accelerating the convergence of the algorithm, and removing the requirement for the FKP.

## Acknowledgements

Part of the work described in this paper was carried out while Alex Clark was a *Professeur Invité* at the University of Marseille.

## References

1. Higuera, C.D.L., Oncina, J.: Inferring deterministic linear languages. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS (LNAI), vol. 2375, pp. 185–200. Springer, Heidelberg (2002)
2. Yokomori, T.: Polynomial-time identification of very simple grammars from positive data. Theoretical Computer Science 298(1), 179–206 (2003)
3. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
4. Marcus, S.: Algebraic Linguistics; Analytical Models. Academic Press, N. Y (1967)
5. Gazdar, G., Klein, E., Pullum, G., Sag, I.: Generalised Phrase Structure Grammar. Basil Blackwell, Malden (1985)
6. Boullier, P.: A Cubic Time Extension of Context-Free Grammars. Grammars 3, 111–131 (2000)
7. Asveld, P.: Generating all permutations by context-free grammars in Chomsky normal form. Theoretical Computer Science 354(1), 118–130 (2006)
8. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
9. Angluin, D.: Queries and concept learning. Mach. Learn. 2(4), 319–342 (1988)
10. Pitt, L.: Inductive inference, dfa's, and computational complexity. LNCS (LNAI), pp. 8–14. Springer, Heidelberg (1989)
11. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27(2), 125–138 (1997)
12. Adriaans, P.: Learning shallow context-free languages under simple distributions. Algebras, Diagrams and Decisions in Language, Logic and Computation 127 (2002)
13. Horning, J.J.: A Study of Grammatical Inference. PhD thesis, Stanford University, Computer Science Department, California (1969)
14. Clark, A.: PAC-learning unambiguous NTS languages. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 59–71. Springer, Heidelberg (2006)
15. Klein, D., Manning, C.: Corpus-based induction of syntactic structure: models of dependency and constituency. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, pp. 478–485 (2004)

# Learning Languages from Bounded Resources: The Case of the DFA and the Balls of Strings[⋆]

Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini

Universities of Lyon, 18 r. Pr. Lauras, F-42000 St-Etienne
{cdlh,janodet,frederic.tantini}@univ-st-etienne.fr

**Abstract.** Comparison of standard language learning paradigms (identification in the limit, query learning, Pac learning) has always been a complex question. Moreover, when to the question of converging to a target one adds computational constraints, the picture becomes even less clear: how much do queries or negative examples help? Can we find good algorithms that change their minds very little or that make very few errors? In order to approach these problems we concentrate here on two classes of languages, the topological balls of strings (for the edit distance) and the deterministic finite automata (Dfa), and (re-)visit the different learning paradigms to sustain our claims.

**Keywords:** Polynomial learnability, deterministic finite automata, balls of strings, edit distance.

## 1 Introduction

The study of the properties of the learning algorithms, particularly those in grammatical inference, can be either empirical (based on experiments from datasets), or theoretical. In the latter, the goal is to study the capacity of the algorithm to retrieve, exactly or approximately, a target language. Often, the goal is also to measure the resources (time, amount of data) necessary to achieve this task. Different paradigms have been proposed to take into account notions of convergence from bounded resources, but none has really imposed itself, and few comparison exists between these definitions.

In this paper, we visit standard criteria for *polynomial* identification and compare them by considering two fundamentally different classes of languages: the regular languages represented by deterministic finite automata, and the balls of strings *w.r.t.* the edit distance [1]. These balls of strings are formed by choosing one specific string, called the centre, and all its neighbours up to a given length for the edit distance, called the radius.

From a practical standpoint, the balls of strings appear in a variety of settings: in approximate string matching tasks, the goal is to find all close matches to some

[⋆] This work was supported in part by the IST Programme of the European Community, under the Pascal Network of Excellence, IST-2006-216886. This publication only reflects the authors' views.

target string [2,3]; in noisy settings, garbled versions of an unidentified string are given and the task is to recover the original string [4]; when using dictionaries, the task can be described as that of finding the intersection between two languages, the dictionary itself and a ball around the target string [5]; in the field of bioinformatics, extracting valid models from large datasets of Dna or proteins can involve looking for substrings at distance less than some given bound, and the set of these approximate substrings can also be represented by balls [6].

When aiming to prove that a class of languages is learnable, there are typically three different settings. The first one, identification in the limit [7], mimes the cognitive process of a child that would acquire his native language by picking up the sentences that are broadcasted in his environment. More formally, information keeps on arriving about a target language and the Learner keeps making new hypotheses. We say that convergence takes place if there is a moment when the process is stationary and the hypothesis is correct.

The second one, query learning [8], looks like a game of riddles where the Learner (pupil) can asks questions (queries) to an Oracle (teacher) about the target language. The game ends when the Learner guesses the target. Of course, the learning results strongly depends on the sort of queries that the Learner is allowed to ask. Both previous paradigms are probably at least as interesting for the negative results they induce as for the positive ones. Indeed, concerning query learning, if a Learner cannot identify an unknown concept by choosing and testing examples, how could he hope to succeed to learn from examples that are imposed by an application?

The last paradigm, Pac learning (for Probably Approximately Correct) [9] is intended to be a more pragmatic setting. It formalizes a situation where one tries to build automatically a predictive model from the data. In this setting, one assumes that there is a (unknown) distribution $\mathcal{D}$ over the strings of the target language, which is used to sample learning and testing examples. Two parameters are fixed: $\epsilon$ is related to the error of the model (*i.e.*, the probability of a string to be misclassified) and $\delta$ is related to the confidence one has in the sampling. Ideally, a good Pac-Learner returns, with high confidence ($> 1 - \delta$), hypotheses that have small error rates ($< \epsilon$).

The three settings are usually difficult to compare, in particular when complexity issues are discussed. Some exceptions are the work by Angluin comparing Pac-learning and using equivalence queries [10], the work by Pitt relating equivalence queries and implicit prediction errors [11], comparisons between learning with characteristic samples, simple Pac [12,13] and Mat in [14]. Other analysis of polynomial aspects of learning grammars, automata and languages can be found in [11,15,16,17]. If the customary comparative approach is to introduce a learning paradigm and survey a variety of classes of languages for this paradigm, we choose here to fix the classes of languages and to proceed to a horizontal analysis of their learnability by visiting the paradigms systematically.

Concerning the Dfa, we complete a long list of known results. Concerning the balls of strings, our results are generally negative: identification in the limit from examples and counter-examples is impossible in most cases, even from membership and equivalence queries. PAC-learning is also impossible in polynomial time, unless $\mathcal{RP} = \mathcal{NP}$. Yet, the errors are usually due to the counter-examples. Hence, we show that it is sometimes (and surprisingly) easier to learn from positive examples only than from positive and negative examples.

Section 2 is devoted to preliminary definitions. In Sections 3, 4 and 5, we focus on the so-called good balls and on the DFA, and we present the results concerning PAC-learning, query learning and polynomial identification in the limit, respectively. We conclude in Section 6.

## 2 Definitions

An *alphabet* $\Sigma$ is a finite nonempty set of symbols called *letters*. In the sequel, we suppose that $|\Sigma| \geq 2$. A *string* $w = a_1 \cdots a_n$ is any finite sequence of letters. We write $\lambda$ for the empty string and $|w|$ for the length of $w$. Let $\Sigma^\star$ denote the set of all strings over $\Sigma$. We say that $u$ is a *subsequence* of $v$, denoted $u \preceq v$, *if*$_{def}$ $u = a_1 \cdots a_n$ and there exist $u_0, \cdots, u_n \in \Sigma^\star$ *s.t.* $v = u_0 a_1 u_1 \cdots a_n u_n$. We introduce the set $lcs(u,v)$ of all longest common subsequences of $u$ and $v$. We also introduce the *hierarchical order*: $u \trianglelefteq v$ *if*$_{def}$ $|u| < |v|$ or ($|u| = |v|$ and $u \leq_{lex} v$), where $\leq_{lex}$ denotes the standard lexicographic order. A *language* is any subset $L \subseteq \Sigma^\star$. Let $\mathbb{N}$ denote the set of non negative integers. For all $k \in \mathbb{N}$, let $\Sigma^{\leq k}$ (respectively $\Sigma^{>k}$) be the set of all strings of length at most $k$ (respectively of length more than $k$). We define $A \oplus B = (A \setminus B) \cup (B \setminus A)$.

Grammatical inference aims at learning the languages of a fixed class $\boldsymbol{\mathcal{L}}$ represented by the grammars of a class $\boldsymbol{\mathcal{G}}$. $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{G}}$ are related by a naming function $\mathbb{L} : \boldsymbol{\mathcal{G}} \to \boldsymbol{\mathcal{L}}$ that is total ($\forall G \in \boldsymbol{\mathcal{G}}, \mathbb{L}(G) \in \boldsymbol{\mathcal{L}}$) and surjective ($\forall L \in \boldsymbol{\mathcal{L}}, \exists G \in \boldsymbol{\mathcal{G}}$ *s.t.* $\mathbb{L}(G) = L$). For any string $w \in \Sigma^\star$ and language $L \in \boldsymbol{\mathcal{L}}$, we shall write $L \models w$ *if*$_{def}$ $w \in L$. Concerning the grammars, they may be understood as any piece of information allowing some parser to recognize the strings. For any string $w \in \Sigma^\star$ and grammar $G \in \boldsymbol{\mathcal{G}}$, we shall write $G \vdash w$ if the parser recognizes $w$. Basically, the parser must be sound and complete *w.r.t.* the semantics: $G \vdash w \iff \mathbb{L}(G) \models w$. In the following, we will mainly consider learning paradigms subject to complexity constraints. In their definitions, $\|G\|$ will denote the size of the grammar $G$ (*e.g.*, the number of states in the case of DFA). Moreover, given a set $X$ of strings, we will write $|X|$ for the cardinality of $X$ and $\|X\|$ for the sum of the lengths of the strings in $X$.

The *edit distance* $d(w,w')$ is the minimum number of *primitive edit operations* needed to transform $w$ into $w'$ [1]. The operation is either (1) a *deletion*: $w = uav$ and $w' = uv$ , or (2) an *insertion*: $w = uv$ and $w' = uav$, or (3) a *substitution*: $w = uav$ and $w' = ubv$, where $u, v \in \Sigma^\star$, $a, b \in \Sigma$ and $a \neq b$. *E.g.*, $d(abaa, aab) = 2$ since $a\underline{b}aa \to aa\underline{a} \to aab$ and the rewriting of $abaa$ into $aab$ cannot be achieved with less than two steps. $d(w,w')$ can be computed in $\mathcal{O}(|w| \cdot |w'|)$ time by dynamic programming [18].

The edit distance is a metric, so we can introduce the *balls* over $\Sigma$. The *ball of centre* $o \in \Sigma^\star$ *and radius* $r \in \mathbb{N}$, denoted $B_r(o)$, is the set of all strings whose distance is at most $r$ from $o$: $B_r(o) = \{w \in \Sigma^\star : d(o,w) \leq r\}$. *E.g.*, if $\Sigma = \{a,b\}$, then $B_1(ba) = \{a,b,aa,ba,bb,aba,baa,bab,bba\}$ and $B_r(\lambda) = \Sigma^{\leq r}$ for all $r \in \mathbb{N}$. We will write $\mathcal{BALL}(\Sigma)$ for the family of all the balls.

To the purpose of grammatical inference, we are going to represent any ball $B_r(o)$ by the pair $(o, r)$ that will play the role of a grammar. Indeed, its size is $|o| + \log r$ (which corresponds to the number of bits necessary to encode the grammar[1]). Moreover, the parser able to decide whether $w \in B_r(o)$ or not is simple: (1) it computes $d(o, w)$ and (2) it checks if this distance is $\leq r$, that can be achieved in time $\mathcal{O}(|o| \cdot |w| + \log r)$. Finally, as $|\Sigma| \geq 2$, we can show that $(o, r)$ is a unique thus *canonical* grammar of $B_r(o)$ [20]. In consequence, we shall also denote by $\mathcal{BALL}(\Sigma)$ the class of grammars associated to the balls.

A ball $B_r(o)$ is called *good* *if*$_{def}$ $r \leq |o|$. The advantage of using good balls is that there is a polynomial relation between the size of the centre and the size of the longest strings in the ball. We will write $\mathcal{GB}(\Sigma)$ for the class of all the good balls (and that of the corresponding grammars).

A *deterministic finite automaton* (DFA) is a 5-tuple $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ *s.t.* $Q$ is a set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states and $\delta : Q \times \Sigma \to Q$ is a transition function. Every DFA can be *completed* with one sink state *s.t.* $\delta$ is a total function. As usual, $\delta$ is extended to $\Sigma^\star$. The *language recognized by* $A$ is $\mathbb{L}(A) = \{w \in \Sigma^\star : \delta(q_0, w) \in F\}$. The size of $A$ is $|Q|$. We will write $\mathcal{DFA}(\Sigma)$ for the class of all DFA over the alphabet $\Sigma$.

The inference of DFA has been intensively studied for forty years at least [7,11,21]. On the other hand, the learnability of the balls is a recent issue [20] motivated by the problem of identifying languages from noisy data. However, our approach raises a preliminary question: if any ball whose grammar would have size $n$ could be recognized by a DFA with $p(n)$ states (for some polynomial $p()$), then one could deduce learnability results on the former from (known) learnability results on the latter. Yet it is generally believed (albeit still an open question) that the transformation of a ball into a DFA is not polynomial [2].

## 3 PAC-Learnability

The PAC paradigm [9] has been widely used in machine learning. It aims at building, with high confidence, good approximations of an unknown concept.

**Definition 1 ($\epsilon$-good hypothesis).** *Let $G$ be the target grammar and $H$ be a hypothesis grammar. Let $\mathcal{D}$ be a distribution over $\Sigma^\star$ and $\epsilon > 0$. We say that $H$ is an $\epsilon$-*good hypothesis *w.r.t. $G$ if*$_{def}$ $Pr_{\mathcal{D}}(x \in \mathbb{L}(G) \oplus \mathbb{L}(H)) < \epsilon$.

The PAC-learnability of grammars from strings of unbounded size has always been tricky [22,16,23]. Indeed, with the standard definition, a PAC-Learner can ask an Oracle to return a sample randomly drawn according to the distribution $\mathcal{D}$. However, in the case of strings, there is always the risk (albeit small) to sample a string too long to account for in polynomial time. In order to avoid this

[1] Notice that $|o| + r$ *is not* a correct measure of the size as implicitly it would mean encoding the radius in unary, something unreasonable [19].

problem, we will sample from a distribution restricted to strings shorter than a specific value given by the following lemma:

**Lemma 1.** *Let $\mathcal{D}$ be a distribution over $\Sigma^\star$. Then $\forall \epsilon, \delta > 0$, with probability at least $1-\delta$, if one draws a sample $X$ of at least $\frac{1}{\epsilon}\ln\frac{1}{\delta}$ strings following $\mathcal{D}$, then the probability of any new string $x$ to be longer than all the strings of $X$ is less than $\epsilon$. Formally, let $\mu_X = \max\{|y| : y \in X\}$, then $Pr_{x\sim\mathcal{D}}(|x| > \mu_X) < \epsilon$.*

*Proof.* Let $\ell$ be the smallest integer *s.t.* $Pr_{\mathcal{D}}(\Sigma^{>\ell}) < \epsilon$. A sufficient condition for $Pr_{\mathcal{D}}(|x| > \mu_X) < \epsilon$ is that we take a sample $X$ large enough to be nearly sure (with probability $> 1-\delta$) to have one string $\geq \ell$. Basically, the probability of drawing $n$ strings in $X$ of length $< \ell$ is $\leq (1-\epsilon)^n$. So the probability of getting at least one string of length $\geq \ell$ is $> 1-(1-\epsilon)^n$. In order to build $X$, we thus need $1-(1-\epsilon)^n > 1-\delta$, that is to say, $(1-\epsilon)^n < \delta$. As $(1-\epsilon)^n \leq e^{-n\epsilon}$, it is sufficient to take $n \geq \frac{1}{\epsilon}\ln\frac{1}{\delta}$ to reach a convenient value for $\mu_X$. □

An algorithm is now asked to learn a grammar given a *confidence* parameter $\delta$ and an *error* parameter $\epsilon$. The algorithm must also be given an upper bound $n$ on the size of the target grammar and an upper bound $m$ on the length of the examples it is going to get (perhaps computed using Lemma 1). The algorithm can query an Oracle for an example randomly drawn according to the distribution $\mathcal{D}$. The query of an example or a counter-example will be denoted Ex(). When the Oracle is only queried for a positive example, we will write Pos-Ex(). And when the Oracle is only queried for string of length $\leq m$, we will write Ex($m$) and Pos-Ex($m$) respectively. Formally, the Oracle will then return a string drawn from $\mathcal{D}$, or $\mathcal{D}(\mathbb{L}(G))$, or $\mathcal{D}(\Sigma^{\leq m})$, or $\mathcal{D}(\mathbb{L}(G) \cap \Sigma^{\leq m})$, respectively, where $\mathcal{D}(L)$ is the restriction of $\mathcal{D}$ to the strings of $L$: $Pr_{\mathcal{D}(L)}(x) = Pr_{\mathcal{D}}(x)/Pr_{\mathcal{D}}(L)$ if $x \in L$, 0 otherwise. $Pr_{\mathcal{D}(L)}(x)$ is not defined if $L = \emptyset$.

**Definition 2 (Polynomial Pac-learnability).** *Let $\boldsymbol{\mathcal{G}}$ be a class of grammars. $\boldsymbol{\mathcal{G}}$ is* Pac-learnable *if$_{def}$ there exists an algorithm $\mathfrak{A}$ s.t. $\forall \epsilon, \delta > 0$, for any distribution $\mathcal{D}$ over $\Sigma^\star$, $\forall n \in \mathbb{N}$, $\forall G \in \boldsymbol{\mathcal{G}}$ of size $\leq n$, for any upper bound $m \in \mathbb{N}$ on the size of the examples, if $\mathfrak{A}$ has access to* Ex*(), $\epsilon$, $\delta$, $n$ and $m$, then with probability $> 1-\delta$, $\mathfrak{A}$ returns an $\epsilon$-good hypothesis w.r.t. $G$. If $\mathfrak{A}$ runs in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $n$ and $m$, we say that $\boldsymbol{\mathcal{G}}$ is* polynomiallly Pac-learnable.

Typical techniques proving non Pac-learnability depend on complexity assumptions [24]. Let us recall that $\mathcal{RP}$ (Randomised Polynomial Time) is the complexity class of decision problems for which a probabilistic Turing machine exists which (1) runs in time polynomial in the input size, (2) on a negative instance, always returns No and (3) on a positive instance, returns Yes with probability $> \frac{1}{2}$ (otherwise, it returns No). The algorithm is randomised: it is allowed to flip a random coin while it is running. The algorithm does not make any error on negative instances, and it is important to remark that on positive instances, since the error is $< \frac{1}{2}$, by repeating the run of the algorithm as many times as necessary, the actual error can be brought to be as small as one wants. We will use the strong belief and assumption that $\mathcal{RP} \neq \mathcal{NP}$ [19].

We are going to show that the good balls are not polynomially PAC-learnable. The proof follows the classical lines for such results: we first prove that the associated consistency problem is $\mathcal{NP}$-hard, through reductions from a well known $\mathcal{NP}$-complete problem (*Longest Common Subsequence*). Then it follows that if a polynomial PAC-learning algorithm for balls existed, this algorithm would provide us with a proof that this $\mathcal{NP}$-complete problem would also be in $\mathcal{RP}$.

**Lemma 2.** *The following problems are $\mathcal{NP}$-complete:*

1. ***Longest Common Subsequence** (LCS): Given $n$ strings $x_1, \dots, x_n$ and an integer $k$, does there exist a string $w$ which is a subsequence of each $x_i$ and is of length $k$?*
2. ***Longest Common Subsequence of Strings of a Given Length** (LCSSGL): Given $n$ strings $x_1, \dots, x_n$ all of length $2k$, does there exist a string $w$ which is a subsequence of each $x_i$ and is of length $k$?*
3. ***Consistent ball** (CB): Given two sets $X_+$ and $X_-$ of strings, does there exist a good ball containing $X_+$ and which does not intersect $X_-$?*

*Proof.* (1) See [25]. (2) See [26, page 42], Problem LCS0. (3) We use a reduction of Problem LCSSGL. We take the strings of length $2k$, and put these with string $\lambda$ into the set $X_+$. We build $X_-$ by taking each string of length $2k$ and inserting every possible symbol once only (hence constructing at most $n(2k+1)|\Sigma|$ strings of size $2k+1$). It follows that a ball that contains $X_+$ but no element of $X_-$ has necessarily a centre of length $k$ and a radius of $k$ (since we focus on good balls only). The centre is then a subsequence of all the strings of length $2k$ that were given. Conversely, if a ball is built using a subsequence of length $k$ as centre, this ball is of radius $k$, contains also $\lambda$, and because of the radius, contains no element of $X_-$. Finally the problem is in $\mathcal{NP}$, since given a centre $o$, it is easy to check if $\max_{x \in X_+} d(o,x) < \min_{x \in X_-} d(o,x)$. □

**Theorem 1.** *Unless $\mathcal{RP} = \mathcal{NP}$, $\boldsymbol{\mathcal{GB}}(\Sigma)$ is not polynomially PAC-learnable.*

*Proof.* Suppose that $\boldsymbol{\mathcal{GB}}(\Sigma)$ is polynomially PAC-learnable with $\mathfrak{A}$ and take an instance $\langle X_+, X_- \rangle$ of Problem CB. We write $h = |X_+| + |X_-|$ and define over $\Sigma^\star$ the distribution $Pr(x) = \frac{1}{h}$ if $x \in X_+ \cup X_-$, 0 if not. Let $\epsilon = \frac{1}{h+1}$, $\delta < \frac{1}{2}$, $m = n = \max\{|w| : w \in X_+\}$. Let $B_r(o)$ be the ball returned by $\mathfrak{A}(\epsilon, \delta, n, m)$ and test if ($X_+ \subseteq B_r(o)$ and $X_- \cap B_r(o) = \emptyset$). If there is no consistent ball, then $B_r(o)$ is inconsistent with the data, so the test is false. If there is a consistent ball, then $B_r(o)$ is $\epsilon$-good, with $\epsilon < \frac{1}{h}$. So, with probability at least $1 - \delta > \frac{1}{2}$, there is no error at all and the test is true. This procedure runs in polynomial time in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $n$ and $m$. So if the good balls were PAC-learnable, there would be a randomized algorithm for the $\mathcal{NP}$-complete CB Problem (by Lemma 2). □

Concerning the PAC-learnability of the DFA, a lot of studies have been done [11,16,23,24] The associated consistency problem is hard [27] and an efficient learning algorithm could be used to invert the RSA encryption function [16]:

**Theorem 2 ([16]).** *$\boldsymbol{\mathcal{DFA}}(\Sigma)$ is not polynomially PAC-learnable.*

In certain cases, it may even be possible to PAC-learn from positive examples only. In this setting, during the learning phase, the examples are sampled following POS-EX() whereas during the testing phase, the sampling is done following EX(), but in both cases the distribution is identical. Again, we can sample using POS-EX($m$), where $m$ is obtained by using Lemma 1 and little additional cost. For any class $\boldsymbol{\mathcal{L}}$ of languages, we get:

**Lemma 3.** *If $\boldsymbol{\mathcal{L}}$ contains 2 languages $L_1$ and $L_2$ s.t. $L_1 \cap L_2 \neq \emptyset$, $L_1 \not\subset L_2$ and $L_2 \not\subset L_1$, then $\boldsymbol{\mathcal{L}}$ is not polynomially* PAC*-learnable from positive examples only.*

*Proof.* Let $w_1 \in L_1 - L_2$, $w_2 \in L_2 - L_1$ and $w_3 \in L_1 \cap L_2$. Consider the distribution $\mathcal{D}_1$ *s.t.* $\Pr_{\mathcal{D}_1}(w_1) = \Pr_{\mathcal{D}_1}(w_3) = \frac{1}{2}$ and the distribution $\mathcal{D}_2$ *s.t.* $\Pr_{\mathcal{D}_2}(w_2) = \Pr_{\mathcal{D}_2}(w_3) = \frac{1}{2}$. Basically, if one learns either $L_1$ from positive examples drawn according to $\mathcal{D}_2$, or $L_2$ from positive examples drawn according to $\mathcal{D}_1$, only the string $w_3$ will be used. However, the error will be $\geq \frac{1}{2}$. □

**Theorem 3.** *(1) $\boldsymbol{\mathcal{GB}}(\Sigma)$ and (2) $\boldsymbol{\mathcal{DFA}}(\Sigma)$ are not polynomially* PAC*-learnable from positive examples only.*

*Proof.* An immediate consequence of Lemma 3 with $L_1 = B_1(a)$, $L_2 = B_1(b)$, $w_1 = aa$, $w_2 = bb$ and $w_3 = ab$. □

## 4 Query Learning

Learning from queries involves the Learner (he) being able to interrogate the Oracle (she) using queries from a given set [8]. The goal of the Learner is to identify a grammar of an unknown language $L$. The Oracle knows $L$ and properly answers to the queries (*i.e.*, she does not lie). Below, we will use three kinds of queries. With the Membership Queries (MQ), the Learner submits a string $w$ to the Oracle and she answers YES if $w \in L$, NO otherwise. With the Equivalence Queries (EQ), he submits (the grammar of) a language $K$ and she answers YES if $K = L$, and a string belonging to $K \oplus L$ otherwise. With the Correction Queries based on the Edit Distance (CQ$_{\text{EDIT}}$), he submits a string $w$ and she answers YES if $w \in L$, and any correction $z \in L$ at minimum edit distance of $w$ otherwise.

**Definition 3.** *A class $\boldsymbol{\mathcal{G}}$ is* polynomially identifiable from queries *if$_{def}$ there is an algorithm $\mathfrak{A}$ able to identify every $G \in \boldsymbol{\mathcal{G}}$ s.t. at any call of a query, the total number of queries and of time used up to that point by $\mathfrak{A}$ is polynomial both in $\|G\|$ and in the size of the information presented up to that point by the Oracle.*

In the case of good balls, we have shown:

**Theorem 4 ([20]).** *(1) $\boldsymbol{\mathcal{GB}}(\Sigma)$ is not polynomially identifiable from* MQ *and* EQ. *(2) $\boldsymbol{\mathcal{GB}}(\Sigma)$ is polynomially identifiable from* CQ$_{\text{EDIT}}$.

Notice however that if the Learner is given one string from a good ball, then he can learn using a polynomial number of MQ only.

Concerning the class of the DFA, we get:

**Theorem 5.** *(1) $\boldsymbol{\mathcal{DFA}}(\Sigma)$ is polynomially identifiable from* MQ *and* EQ *[21], but is not from (2)* MQ *only [8], nor (3)* EQ *only [10], nor (4)* $\text{CQ}_{\text{EDIT}}$ *only.*

*Proof.* (4) Let $\mathcal{A}_w$ denote the DFA that recognizes $\Sigma^\star \setminus \{w\}$. Let $n \in \mathbb{N}$ and $\boldsymbol{\mathcal{DFA}}_{\leq n} = \{\mathcal{A}_w : w \in \Sigma^{\leq n}\}$. Following [10], we describe an Adversary that maintains a set $X$ of all the possible DFA. At the beginning, $X = \boldsymbol{\mathcal{DFA}}_{\leq n}$. Each time the correction of any string $w$ is demanded, the Adversary answers YES and eliminates only one DFA of $X$: $\mathcal{A}_w$. As there is $\Omega(|\Sigma|^n)$ DFA in $\boldsymbol{\mathcal{DFA}}_{\leq n}$, identifying one of them will require $\Omega(|\Sigma|^n)$ queries in the worst case. □

## 5 Polynomial Identification in the Limit

Identification in the limit [7] is standard: a Learner receives an infinite sequence of information (presentation) that should help him to find the grammar $G \in \boldsymbol{\mathcal{G}}$ of an unkown target language $L \in \boldsymbol{\mathcal{L}}$. The set of admissible presentations is denoted by **Pres**, each presentation being a function $\mathbb{N} \to X$ where $X$ is any set. Given $\boldsymbol{f} \in$ **Pres**, we will write $\boldsymbol{f}_m$ for the $m+1$ first elements of $\boldsymbol{f}$, and $\boldsymbol{f}(n)$ for its $n^{th}$ element. Below, we will consider two sorts of presentations. When **Pres**=TEXT, all the strings in $L$ are presented: $\boldsymbol{f}(\mathbb{N}) = \mathbb{L}(G)$. When **Pres**=INFORMANT, a presentation is of labelled pairs $(w, l)$ where $(w \in L \Rightarrow l = +)$ and $(w \notin L \Rightarrow l = -)$: $\boldsymbol{f}(\mathbb{N}) = \mathbb{L}(G) \times \{+\} \cup \overline{\mathbb{L}(G)} \times \{-\}$; we will write **Pres** = PRESENTATION for all the results that concern both TEXT and INFORMANT.

**Definition 4.** *We say that $\boldsymbol{\mathcal{G}}$ is* identifiable in the limit from **Pres** *$if_{def}$ there exists an algorithm $\mathfrak{A}$ s.t. for all $G \in \boldsymbol{\mathcal{G}}$ and for any presentation $\boldsymbol{f}$ of $\mathbb{L}(G)$, there exists a rank $n$ s.t. for all $m \geq n$, $\mathfrak{A}(\boldsymbol{f}_m) = \mathfrak{A}(\boldsymbol{f}_n)$ and $\mathbb{L}(\mathfrak{A}(\boldsymbol{f}_n)) = \mathbb{L}(G)$.*

This definition yields a number of learnability results. However, the absence of efficiency constraints often leads to unusable algorithms. Firstly, it seems reasonable that the amount of time an algorithm has to learn should be bounded:

**Definition 5 (Polynomial Update Time).** *An algorithm $\mathfrak{A}$ is said to have* polynomial update time *$if_{def}$ there is a polynomial $p()$ s.t., for every presentation $\boldsymbol{f}$ and every integer $n$, computing $\mathfrak{A}(\boldsymbol{f}_n)$ requires $\mathcal{O}(p(\|\boldsymbol{f}_n\|))$ time.*

It is known that polynomial update time is not sufficient [11]: a Learner could receive an exponential number of examples without doing anything but wait, and then use the amount of time he saved to solve any $\mathcal{NP}$-hard problem... Polynomiality should also concern the minimum amount of data that any Learner requires:

**Definition 6 (Polynomial Characteristic Sample).** *We say that $\boldsymbol{\mathcal{G}}$ admits* polynomial characteristic samples *$if_{def}$ there exist an algorithm $\mathfrak{A}$ and a polynomial $p()$ s.t. for all $G \in \boldsymbol{\mathcal{G}}$, there exists* CS $\subseteq X$ *s.t. (1)* $\|\text{CS}\| \leq p(\|G\|)$*, (2)* $\mathbb{L}(\mathfrak{A}(\text{CS})) = \mathbb{L}(G)$ *and (3) for all $\boldsymbol{f} \in$ **Pres**, for all $n \geq 0$, if* $\text{CS} \subseteq \boldsymbol{f}_n$ *then* $\mathfrak{A}(\boldsymbol{f}_n) = \mathfrak{A}(\text{CS})$*. Such a set* CS *is called a* characteristic sample *of $G$ for $\mathfrak{A}$. If $\mathfrak{A}$ exists, we say that $\boldsymbol{\mathcal{G}}$* is identifiable in the limit in CS polynomial time*.*

Lastly, polynomiality may concern either the number of implicit prediction errors [11] or the number of mind changes (Mc) [28] done by the learner:

**Definition 7 (Implicit Prediction Errors).** *We say that an algorithm $\mathfrak{A}$ makes an* implicit prediction error *(*Ipe*) at time $n$ of a presentation $\boldsymbol{f}$ if$_{def}$ $\mathfrak{A}(\boldsymbol{f}_{n-1}) \not\vdash \boldsymbol{f}(n)$. $\mathfrak{A}$ is called* consistent *if$_{def}$ it changes its mind each time a prediction error is detected with the new presented element.*

*$\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit in* Ipe *polynomial time if$_{def}$ (1) $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit, (2) $\mathfrak{A}$ has polynomial update time and (3) $\mathfrak{A}$ makes a polynomial number of implicit prediction errors: let $\#\text{Ipe}(\boldsymbol{f}) = |\{k \in \mathbb{N} : \mathfrak{A}(\boldsymbol{f}_k) \not\vdash \boldsymbol{f}(k+1)\}|$; there exists a polynomial $p()$ s.t. for each $G \in \boldsymbol{\mathcal{G}}$ and each presentation $\boldsymbol{f}$ of $\mathbb{L}(G)$, $\#\text{Ipe}(\boldsymbol{f}) \leq p(\|G\|)$.*

**Definition 8 (Mind Changes).** *We say that an algorithm $\mathfrak{A}$* changes its mind *(*Mc*) at time $n$ of presentation $\boldsymbol{f}$ if$_{def}$ $\mathfrak{A}(\boldsymbol{f}_n) \neq \mathfrak{A}(\boldsymbol{f}_{n-1})$. $\mathfrak{A}$ is called* conservative *if$_{def}$ it never changes its mind when the current hypothesis is consistent with the new presented element.*

*$\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit in* Mc *polynomial time if$_{def}$ (1) $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit, (2) $\mathfrak{A}$ has polynomial update time and (3) $\mathfrak{A}$ makes a polynomial number of mind changes: Let $\#\text{Mc}(\boldsymbol{f}) = |\{k \in \mathbb{N} : \mathfrak{A}(\boldsymbol{f}_k) \neq \mathfrak{A}(\boldsymbol{f}_{k+1})\}|$; there exists a polynomial $p()$ s.t. for each $G \in \boldsymbol{\mathcal{G}}$ and each presentation $\boldsymbol{f}$ of $\mathbb{L}(G)$, $\#\text{Mc}(\boldsymbol{f}) \leq p(\|G\|)$.*

Concerning both last notions, one can notice that if an algorithm $\mathfrak{A}$ is consistent then $\#\text{Ipe}(\boldsymbol{f}) \leq \#\text{Mc}(\boldsymbol{f})$ for every presentation $\boldsymbol{f}$. Likewise, if $\mathfrak{A}$ is conservative then $\#\text{Mc}(\boldsymbol{f}) \leq \#\text{Ipe}(\boldsymbol{f})$. So we deduce the following lemma:

**Lemma 4.** *If $\mathfrak{A}$ identifies the class $\boldsymbol{\mathcal{G}}$ in* Mc *polynomial time and is consistent, then $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in* Ipe *polynomial time. Conversely, if $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in* Ipe *polynomial time and is conservative, then $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in* Mc *polynomial time.*

## 5.1 Polynomial Identification from Text

The aim of this section is to show the following result:

**Theorem 6.** *$\boldsymbol{\mathcal{GB}}(\Sigma)$ is identifiable in the limit from* Text *in (1)* Mc *polynomial time, (2)* Ipe *polynomial time and (3)* Cs *polynomial time.*

Notice that as the Dfa recognize a *superfinite* class of languages (*i.e.*, containing all the finite languages and at least one infinite language), it is impossible to identify the class in the limit from positive examples only:

**Theorem 7 ([7]).** *$\boldsymbol{\mathcal{DFA}}(\Sigma)$ is not identifiable in the limit from* Text.

In order to prove Theo. 6, we will need to build the minimum consistent good ball containing a set $X = \{x_1, \ldots, x_n\}$ of strings (sample). This problem is $\mathcal{NP}$-hard but some instances are efficiently solvable. Let $X^{max}$ (*resp.* $X^{min}$) denote the set of all longest (*resp.* shortest) strings of $X$. A *minimality fingerprint* is a subset $\{u, v, w\} \subseteq X$ *s.t.* (1) $u, v \in X^{max}$, (2) $w \in X^{min}$, (3) $|u| - |w| = 2r$ for some $r \in \mathbb{N}$, (4) $u$ and $v$ have only one longest common subsequence, that is, $lcs(u,v) = \{o\}$ for some $o \in \Sigma^\star$, (5) $|o| = |u| - r$ and (6) $X \subseteq B_r(o)$.

Checking if $X$ contains a minimality fingerprint, and computing $o$ and $r$ can be achieved in polynomial time (in $\|X\|$). Indeed, the only critical point is that the cardinal of $lcs(u,v)$ may be $> 1.442^n$ [29] (where $n = |u| = |v|$); nevertheless, a data structure such as the Lcs-graph [30] allows one to conclude polynomially. Moreover, the minimality fingerprints are meaningful. Indeed, only the properties of the edit distance are needed to show that if $X$ contains a minimality fingerprint $\{u,v,w\}$ for the ball $B_r(o)$ and $X \subseteq B_{r'}(o')$, then either $r' > r$, or ($r' = r$ and $o' = o$). In other words, $B_r(o)$ is the smallest ball containing $X$ *w.r.t.* the radius.

We can now consider Algo. 1. This algorithm does identify $\mathcal{GB}(\Sigma)$ in the limit since if $B_r(o)$ denotes the target ball, then at some point, the algorithm will meet the strings $u = a^r o, v = b^r o$, and some $w$ of length $|o| - r$ that constitute a minimality fingerprint for $B_r(o)$. Moreover, it obviously has a polynomial update time. Finally, it makes a polynomial number of Mc. Indeed, it only changes its hypothesis in favour of a valid ball if the ball has a larger radius than all the valid balls it has ever conjecture, that may happen $\leq r$ times. And it only changes its hypothesis in favour of a junk ball if either it must abandon a valid ball, or if the actual junk ball does not contain all the examples, that may happen $\leq r + 2r$ times. So the total number of Mc is $\leq 4r$. So Claim (1) holds.

Concerning Claim (2), note that Algo. 1 is consistent (thanks to the use of the junk balls), thus Claim (2) holds by Lemma 4. Lastly, every minimality fingerprint is a characteristic set that makes Algo. 1 converge, so Claim (3) holds.

**Algorithm 1.** Identification of good balls from text.

```
Data: A text $\boldsymbol{f} = \{x_1, x_2, \ldots\}$
read($x_1$); $c \leftarrow x_1$; output ($x_1, 0$);
while true do
    read($x_i$);
    if $\boldsymbol{f}_i$ is a minimality fingerprint for $B_r(o)$ then
        output $(o, r)$ (* valid ball *)
    else
        if $c \notin \boldsymbol{f}_i^{max}$ then $c \leftarrow$ any string in $\boldsymbol{f}_i^{max}$;
        output $(c, |c|)$ (* junk ball *)
    end
end
```

### 5.2 Polynomial Identification from Informant

**Theorem 8.** *(1) $\mathcal{GB}(\Sigma)$ is not identifiable from* Informant *in* Ipe *polynomial time, but is identifiable in (2)* Mc *polynomial time and (3)* Cs *polynomial time.*

*Proof.* (1) Similar proof as that of [11] for the Dfa: if $\mathcal{GB}(\Sigma)$ was identifiable in Ipe polynomial time from Informant, then $\mathcal{GB}(\Sigma)$ would be polynomially identifiable from Eq, that contradicts Theo. 4. (2) As the hypotheses are not necessarily consistent with the data, one can use Algo. 1, ignoring the negative examples. (3) Same characteristic sets as those of Theo. 6, Claim (3). □

**Theorem 9.** *(1) $\mathcal{DFA}(\Sigma)$ is not identifiable from* Informant *in* Ipe *polynomial time [11], but is identifiable in (2)* Mc *polynomial time and (3)* Cs *polynomial time [31,32].*

Let us prove Claim (2) with minimality fingerprints again. We say that $X = \langle X_+, X_- \rangle$ contains a *minimality fingerprint* $if_{def}$ the following conditions hold: (1) let $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ be the Dfa computed by Rpni [32] on $X$ possibly completed with one hole state; (2) for all $q \in Q$, the smallest string $w_q$ *w.r.t.* the hierarchical order $\trianglelefteq$ *s.t.* $\delta(q_0, w_q) = q$ belongs to either $X_+$ if $q \in F$, or $X_-$ if $q \notin F$; (3) for all $q \in Q, a \in \Sigma$, $w_q a$ belongs to either $X_+$ if $\delta(q,a) \in F$, or $X_-$ if $\delta(q,a) \notin F$; (4) for all $p, q, r \in Q, a \in \Sigma$ *s.t.* $\delta(p,a) = q \neq r$, there exists $f \in \Sigma^\star$ *s.t.* either $(w_p af \in X_+, w_r f \in X_-)$, or $(w_p af \in X_-, w_r f \in X_+)$.

Notice that not all the Dfa have minimality fingerprints. Moreover, every fingerprint contains a characteristic sample of $A$ for Rpni [32], plus new information: actually, *all* the states and the transitions of $A$ are determined by the fingerprint, so any other complete Dfa $A'$ compatible with $X$ necessarily has more states than $A$. $A$ is thus the unique complete minimal Dfa compatible with $X$. Lastly, checking if $X$ contains a minimality fingerprint, and computing $A$ is achievable in polynomial time (in $\|X\|$).

We can now define a Learner $\mathfrak{A}$. At each step, $\mathfrak{A}$ tests if $\boldsymbol{f}_i$ contains a minimality fingerprint. If yes, $\mathfrak{A}$ changes its mind in favour of the Dfa $A_i$ returned by Rpni on $\boldsymbol{f}_i$. If no, $\mathfrak{A}$ returns the previous hypothesis (that may not be consistent). Clearly, the number of states of $A_i$ strictly increases (thanks to the fingerprints). As this number is bounded by the number of states of the target Dfa $A$, we get $\#\text{Mc}(\boldsymbol{f}) \leq \|A\|$. Moreover, at some point, a fingerprint (thus a characteritic set of $A$) will appear in the data, and then $\mathfrak{A}$ will converge.

## 6 Conclusion

In this paper, we have performed a systematic study of two classes of languages whose definitions is based on very different principles. Table 1 sumarize our results. Those marked with a $^\dagger$ were proved in this article.

Clearly, the goal of this work was not to show that any paradigm is equivalent or better than any other: comparing two classes is not sufficient. Nevertheless, we have shown that several hints were wrong. For instance, it is wrong to think that the identification in Mc polynomial time implies the identification in Ipe polynomial time. It is also wrong to think that it is easier to learn from positive and negative examples (Informant) than from positive examples only (Text) (because in some paradigm, misclassifying negative examples is expensive in terms of complexity).

In Table 1, we also show (without proof, due to the lack of space), the results that concern all the balls including those that are not good. Let us recall that a ball $B_r(o)$ is good $if_{def}$ $r \leq |o|$, so for a bad ball, it is possible that $r \gg 2^{|o|}$. In

**Table 1.** A synthetic view of the results presented in this paper. † marks the theorems proved above. Due to the lack of space, we just claim the results concerning the general balls of $\mathcal{BALL}(\Sigma)$.

| Criterion | $\mathcal{GB}(\Sigma)$ | | $\mathcal{DFA}(\Sigma)$ | | $\mathcal{BALL}(\Sigma)$ |
|---|---|---|---|---|---|
| Pac Inform. | No† | Theo. 1 | No | Theo. 2 | No |
| Pac Text | No† | Theo. 3 (1) | No† | Theo. 3 (2) | No |
| Ipe Inform. | No† | Theo. 8 (1) | No | Theo. 9 (1) | No |
| Ipe Text | Yes† | Theo. 6 (2) | No | Theo. 7 | No |
| Mc Inform. | Yes† | Theo. 8 (2) | Yes† | Theo. 9 (2) | Yes |
| Mc Text | Yes† | Theo. 6 (1) | No | Theo. 7 | No |
| Cs Inform. | Yes† | Theo. 8 (3) | Yes | Theo. 9 (3) | No |
| Cs Text | Yes† | Theo. 6 (3) | No | Theo. 7 | No |
| Mq (or Eq) | No | Theo. 4 (1) | No | Theo. 5 (2,3) | No |
| Mq and Eq | No | Theo. 4 (1) | Yes | Theo. 5 (1) | No |
| $\text{Cq}_{\text{Edit}}$ | Yes | Theo. 4 (2) | No | Theo. 5 (4) | No |

this case, the longest strings of $B_r(o)$, whose length is $|o| + r$, which delimitate the upper boarder of $B_r(o)$, are not polynomially related to the size of the ball ($|o| + \log r$). So the picture is the same as that of the *non deterministic automata* for which one has to consider strings of exponential length in order to distinguish two states [17,33]. Hence, studying the learnability of all the balls is a way to explore the limits of the paradigms, reached when an algorithm cannot get round of exponential strings anymore.

Finally, one can note that the good balls are not learnable from a polynomial number of Mq and Eq, that is the case of the Dfa. As the balls are finite languages, they are recognizable with Dfa. Thus a subclass of a learnable class could be non learnable! We conjecture, following [5,2], that this is because the size of the minimal Dfa recognizing a ball is exponential in the size of the ball.

## References

1. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR 163(4), 845–848 (1965)
2. Navarro, G.: A guided tour to approximate string matching. ACM computing surveys 33(1), 31–88 (2001)
3. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Computing Survey 33(3), 273–321 (2001)
4. Kohonen, T.: Median strings. Pattern Recognition Letters 3, 309–313 (1985)
5. Schulz, K.U., Mihov, S.: Fast string correction with Levenshtein automata. Int. Journal on Document Analysis and Recognition 5(1), 67–85 (2002)
6. Sagot, M.F., Wakabayashi, Y.: Pattern inference under many guises. In: Recent Advances in Algorithms and Combinatorics, pp. 245–287. Springer, Heidelberg (2003)

7. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
8. Angluin, D.: Queries and concept learning. Machine Learning Journal 2, 319–342 (1987)
9. Valiant, L.G.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)
10. Angluin, D.: Negative results for equivalence queries. Machine Learning Journal 5, 121–150 (1990)
11. Pitt, L.: Inductive inference, DFA's, and computational complexity. In: Jantke, K.P. (ed.) AII 1989. LNCS, vol. 397, pp. 18–44. Springer, Heidelberg (1989)
12. Li, M., Vitanyi, P.: Learning simple concepts under simple distributions. Siam Journal of Computing 20, 911–935 (1991)
13. Denis, F.: Learning regular languages from simple positive examples. Machine Learning Journal 44(1), 37–66 (2001)
14. Parekh, R.J., Honavar, V.: On the relationship between models for learning in helpful environments. In: Oliveira, A.L. (ed.) ICGI 2000. LNCS (LNAI), vol. 1891, pp. 207–220. Springer, Heidelberg (2000)
15. Haussler, D., Kearns, M.J., Littlestone, N., Warmuth, M.K.: Equivalence of models for polynomial learnability. Information and Computation 95(2), 129–161 (1991)
16. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. In: 21st ACM Symposium on Theory of Computing (STOC 1989), pp. 433–444 (1989)
17. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning Journal 27, 125–138 (1997)
18. Wagner, R., Fisher, M.: The string-to-string correction problem. Journal of the ACM 21, 168–178 (1974)
19. Papadimitriou, C.M.: Computational Complexity. Addison Wesley, New York (1994)
20. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Learning balls of strings with correction queries. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 18–29. Springer, Heidelberg (2007)
21. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Control 39, 337–350 (1987)
22. Warmuth, M.: Towards representation independence in PAC-learning. In: Jantke, K.P. (ed.) AII 1989. LNCS, vol. 397, pp. 78–103. Springer, Heidelberg (1989)
23. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
24. Pitt, L., Valiant, L.G.: Computational limitations on learning from examples. Journal of the ACM 35(4), 965–984 (1988)
25. Maier, D.: The complexity of some problems on subsequences and supersequences. Journal of the ACM 25, 322–336 (1977)
26. de la Higuera, C., Casacuberta, F.: Topology of strings: Median string is NP-complete. Theoretical Computer Science 230, 39–48 (2000)
27. Pitt, L., Warmuth, M.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the ACM 40(1), 95–142 (1993)
28. Angluin, D., Smith, C.: Inductive inference: theory and methods. ACM computing surveys 15(3), 237–269 (1983)
29. Greenberg, R.I.: Bounds on the number of longest common subsequences. Technical report, Loyola University (2003), `http://arXiv.org/abs/cs/0301030v2`

30. Greenberg, R.I.: Fast and simple computation of all longest common subsequences. Technical report, Loyola University (2002), http://arXiv.org/abs/cs.DS/0211001
31. Gold, E.M.: Complexity of automaton identification from given data. Information and Control 37, 302–320 (1978)
32. Oncina, J., García, P.: Identifying regular languages in polynomial time. In: Advances in Structural and Syntactic Pattern Recognition. Series in Machine Perception and Artificial Intelligence, vol. 5, pp. 99–108. World Scientific, Singapore (1992)
33. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using RFSA. Theoretical Computer Science 313(2), 267–294 (2004)

# Relevant Representations for the Inference of Rational Stochastic Tree Languages[*]

François Denis[1], Édouard Gilbert[2], Amaury Habrard[1], Faïssal Ouardi[1], and Marc Tommasi[2]

[1] Laboratoire d'Informatique Fondamentale, CNRS, Aix-Marseille Université
`{francois.denis,amaury.habrard,faissal.ouardi}@lif.univ-mrs.fr`
[2] Laboratoire d'Informatique Fondamentale de Lille (L.I.F.L.), INRIA and É.N.S. Cachan
`{edouard.gilbert,marc.tommasi}@inria.fr`

**Abstract.** Recently, an algorithm - DEES- was proposed for learning rational stochastic tree languages. Given a sample of trees independently and identically drawn according to a distribution defined by a rational stochastic language, DEES outputs a linear representation of a rational series which converges to the target. DEES can then be used to identify in the limit with probability one rational stochastic tree languages. However, when DEES deals with finite samples, it often outputs a rational tree series which does not define a stochastic language. Moreover, the linear representation can not be directly used as a generative model. In this paper, we show that any representation of a rational stochastic tree language can be transformed in a reduced normalised representation that can be used to generate trees from the underlying distribution. We also study some properties of consistency for rational stochastic tree languages and discuss their implication for the inference. We finally consider the applicability of DEES to trees built over an unranked alphabet.

## 1 Introduction

In this paper, we consider the problem of learning probability distribution over trees from a sample of trees independently and identically distributed (i.i.d.), in a given class of models. In this context, the learning process has two main objectives: Finding the correct structure of the representation and estimating precisely the parameters of the model. Because we adopt a machine learning standpoint, we restrict ourselves to classes of probabilistic languages that can be somehow finitely presented. Probabilistic tree automata (PTA) are a usual representations for rational stochastic tree languages (RSTL). In a PTA, each rule is equipped with a weight in $[0, 1]$ and a per state normalisation is imposed. Nonetheless, a first drawback is that it may be not decidable to know whether a PTA is consistent *i.e.* whether it represents a probability distribution on trees.

[*] This work was partially supported by the Atash project ANR-05-RNTL00102 and the Marmota project ANR-05-MMSA-0016.

One difficulty comes from the fact that a RSTL may be such that the average size of trees may be undefined. A second drawback of PTA is that they admit no canonical representation. Thus, most of learning algorithms approaches based on grammatical inference fail for the class of PTA.

Recent approaches have proposed to work in a larger class of representation: The class of rational stochastic tree languages that can be represented under a linear form of a tree series. The models of this class can be equivalently representing by weighted tree automata with parameters in $\mathbb{R}$ (hence with weights that can be negative and without any per state normalisation condition). This class has two interesting properties: It has a high level of expressiveness since it strictly includes the class of RSTL and it admits a canonical form with a minimal number of parameters. Based on these properties, linear representations of RSTL are a good candidate from a grammatical inference standpoint. A recent algorithm, DEES, able to identify in the limit with probability one the class of rational stochastic tree languages RSTL was proposed in [1]. However, this algorithm has two main drawbacks when working with finite samples. It often outputs a rational tree series that does not define a stochastic language, and the representation of the series can not be directly used as a generative model. This comes from the fact that the canonical representation is more adapted for finding the structure of the model and estimating the parameters. We do not obtain a representation of a probability distribution that factorises into a product of probabilities associated with each state. When we need a generative model, we claim that we have to use another representation. Our first contribution is to show that any canonical representation of a rational stochastic tree language admits a normalised reduced representation of the same size which can be easily used in a generative process. Then, we examine some conditions of consistency for rational stochastic languages. Indeed, as for probabilistic context-free grammars [2,3], the consistency can not be ensured only with syntactical properties. We discuss then the influence of these conditions to the problem of inferring rational stochastic tree languages. We finish by studying the applicability of our approach to trees that are built from an unranked alphabet. Actually, a bijection can be made between the unranked representation and a ranked one, allowing us to apply our algorithm to the unranked case.

The paper is organized as follows. Definitions and notations are presented in Section 2. Section 3 deals with the normalised reduced representation of rational stochastic tree language. The consistency conditions are evoked in Section 4. The paper terminates by Section 5 on unranked trees.

## 2 Preliminaries

In this section, we recall definitions of trees, (rational) tree series, weighted automata and (rational) stochastic tree languages. We mainly follow notations and definitions from [4] about trees and tree automata. Formal power tree series have been introduced in [5] where the main results appear.

**Trees and Contexts.** Let $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_p$ be a ranked alphabet where the elements in $\mathcal{F}_m$ are the function symbols of rank $m$. Let $\mathcal{X}$ be a countable set of variables. The set $T(\mathcal{F}, \mathcal{X})$ is the smallest set satisfying: $\mathcal{F}_0 \cup \mathcal{X} \subseteq T(\mathcal{F}, \mathcal{X})$, for $f \in \mathcal{F}_m,\ m \geq 1$, and $t_1, \ldots, t_m \in T(\mathcal{F}, \mathcal{X})$, $f(t_1, \ldots, t_m) \in T(\mathcal{F}, \mathcal{X})$.

We call *trees*, elements in $T(\mathcal{F}, \emptyset) = T(\mathcal{F})$. For any tree $t$, let us denote by $|t|_f$ the number of occurrences of the symbol $f \in \mathcal{F}$ in $t$ and by $|t|$, the size $\sum_{f \in \mathcal{F}} |t|_f$ of $t$. The *height* of a tree $t$ is defined by: $\text{height}(t) = 0$ if $t \in \mathcal{F}_0$ and $\text{height}(t) = 1 + \max\{\text{height}(t_i) | i = 1..m\}$ if $t = f(t_1, \ldots, t_m)$. We suppose given a total order $\leq$ on $T(\mathcal{F})$ which satisfies $\text{height}(t) < \text{height}(s) \Rightarrow t < s$.

*Contexts* are elements $c$ of $C_n(\mathcal{F}) \subset T(\mathcal{F}, \mathcal{X})$ where $n$ distinct variables $\$_1, \ldots \$_n$ appears exactly once in $c$. Let $c$ be a context in $C_n(\mathcal{F})$ and $t_1, \ldots, t_n$ be trees. In the following, the notation $c[\$_1 \leftarrow t_1, \ldots, \$_n \leftarrow t_n]$ or simply $c[t_1, \ldots, t_m]$ represents the tree that results from substituting the $\$_i$'s by the $t_i$'s in $c$. $C_1(\mathcal{F})$ is simply denoted by $C(\mathcal{F})$. We say that a set $A$ is *prefixial* whenever for any $c \in C(\mathcal{F})$ and any $t \in T(\mathcal{F})$, $c[t] \in A \Rightarrow t \in A$.

**Formal Power Tree Series.** A *(formal power) tree series* on $T(\mathcal{F})$ is a mapping $r : T(\mathcal{F}) \to \mathbb{R}$. The vector space of all tree series on $T(\mathcal{F})$ is denoted by $\mathbb{K}\langle\langle\mathcal{F}\rangle\rangle$.

Let $V$ be a finite dimensional vector space over $\mathbb{R}$. We denote by $\mathcal{L}(V^p; V)$ the set of $p$-linear mappings from $V^p$ to $V$. Let $\mathcal{L} = \cup_{p \geq 0} \mathcal{L}(V^p; V)$. We denote by $V^*$ the dual space of $V$, *i.e.* the vector space composed of all the linear forms defined on $V$.

*A linear representation* of $T(\mathcal{F})$ is a couple $(V, \mu)$, where $V$ is a finite dimensional vector space over $\mathbb{R}$, and where $\mu : \mathcal{F} \to \mathcal{L}$ maps $\mathcal{F}_p$ into $\mathcal{L}(V^p; V)$ for each $p \geq 0$. Thus for each $f \in \mathcal{F}_p,\ \mu(f) \colon V^p \to V$ is $p$-linear. Function $\mu$ extends uniquely to a morphism $\mu \colon T(\mathcal{F}) \to V$ by: $\mu(f(t_1, \ldots, t_p)) = \mu(f)(\mu(t_1), \ldots, \mu(t_p))$. Let $V_{T(\mathcal{F})}$ be the vector subspace of $V$ spanned by $\mu(T(\mathcal{F}))$: $(V_{T(\mathcal{F})}, \mu)$ is a linear representation of $T(\mathcal{F})$.

Let $r$ be a tree series over $T(\mathcal{F})$, $r$ is said to be *recognizable* if there exists a triple $(V, \mu, \lambda)$, where $(V, \mu)$ is a linear representation of $T(\mathcal{F})$, and $\lambda : V \to \mathbb{R}$ is a linear form, such that $r(t) = \lambda(\mu(t))$ for all $t$ in $T(\mathcal{F})$. The triple $(V, \mu, \lambda)$ is called a *linear representation for r*.

It has been shown in [6] that the notions of recognizable tree series and rational tree series (*i.e.* tree series characterized by rational tree expressions) coincide. From now on, we shall refer to them by using the term of *rational* tree series.

**Definition 1.** *A* stochastic tree language *over $T(\mathcal{F})$ is a tree series $r \in \mathbb{R}\langle\langle\mathcal{F}\rangle\rangle$ such that for any $t \in T(\mathcal{F})$, $0 \leq r(t) \leq 1$ and $\sum_{t \in T(\mathcal{F})} r(t) = 1$. Let $p$ be a stochastic language, let $c \in C(\mathcal{F})$ be such that there exists a tree $t$ such that $p(c[t]) \neq 0$. We define the stochastic language $c^{-1}p$ by $c^{-1}p(t) = \frac{p(c[t])}{\sum_{t' \in T(\mathcal{F})} p(c[t'])}$.*

A *rational stochastic tree language* (RSTL) is a stochastic tree language which admits a linear representation. The set of rational stochastic tree languages is denoted by $\mathcal{S}^{rat}(\mathcal{F})$.

**Weighted Tree Automata.** A *weighted tree automaton*[1] (WTA) over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, \tau, \delta)$ where $Q$ is a set of states, $\tau$ is a mapping from $Q$ to $\mathbb{R}$ and $\delta$ is a mapping from $\cup_{m\geq 0}\mathcal{F}_m \times Q^m \times Q$ to $\mathbb{R}$. The mapping $\delta$ can be interpreted as a set $\Delta$ of rules which can be written in a bottom-up or a top-down way:

$f(q_1, \dots, q_m) \xrightarrow{w} q \in \Delta$(or $q \xrightarrow{w} f(q_1, \dots, q_m) \in \Delta$) iff $\delta(f, q_1, \dots, q_m, q) = w \wedge w \neq 0$.

The weight $w$ of a rule $r$ is denoted by $w(r)$. For any $q \in Q$, we denote by $\Delta_q$ the subset of $\delta$ composed of the (top-down) rules whose lhs is $q$ and by $\Delta_{f,q}$ the subset of $\Delta_q$ composed of rules containing the symbol $f \in \mathcal{F}$ in the rhs. A series $r_q$ can be associated with any state $q$ by: $r_q(f(t_1, \dots, t_n)) = \sum_{r\in\Delta_q} w(r) \prod_{i=1}^n r_{q_i}(t_i)$. Then the WTA $\mathcal{A}$ computes the series $r$ defined by: $r(t) = \sum_{q\in Q} \tau(q) r_q(t)$.

WTA and linear representations are two equivalent ways to represent rational series. For example, let $(V, \mu, \lambda)$ be a linear representation of the tree series $r \in \mathbb{R}\langle\langle\mathcal{F}\rangle\rangle$ and let $B = \{e_1, \dots, e_n\}$ be a basis of $V$. A WTA $\mathcal{A} = (Q, \mathcal{F}, \lambda, \delta)$ can be associated with $(V, \mu, \lambda, B)$ where $Q = \{e_1, \dots, e_n\}$, and $\delta(f, e_{i_1}, \dots, e_{i_m}, e_j) = w_j$ for any $f \in \mathcal{F}_m$ where $\mu(f)(e_{i_1}, \dots, e_{i_m}) = \sum_j w_j e_j$. Conversely, an equivalent linear representation can be associated with any weighted tree automaton (see Example 1 below).

Note here that a *probabilistic tree automaton* (PTA) is a specific case of WTA $\mathcal{A} = (Q, \mathcal{F}, \tau, \delta)$ satisfying the following conditions: (i) $\delta$ and $\tau$ take their values in $[0, 1]$, (ii) $\sum_{q\in Q} \tau(q) = 1$, (iii) for any $q \in Q$, $\sum_{r\in\Delta_q} w(r) = 1$.

It can be shown that any PTA computes a rational tree series $r$ that satisfies $r(t) \geq 0$ for any tree $t$ and $\sum_t r(t) \leq 1$.

It can be shown that there exist rational stochastic tree languages that cannot be computed by any probabilistic automaton (see [7] for an example in the case of word stochastic languages).

*Example 1. A* WTA *representing a rational stochastic tree language.* Let $\mathcal{A} = (Q, \mathcal{F}, \tau, \delta)$ be the WTA defined by $Q = \{q_1, q_2\}$, $\mathcal{F} = \{a, f(\cdot, \cdot)\}$, $\tau(q_1) = 2, \tau(q_2) = -1$ and $\Delta = \{q_1 \xrightarrow{2/3} a, q_1 \xrightarrow{1/3} f(q_1, q_1), q_2 \xrightarrow{3/4} a, q_2 \xrightarrow{1/4} f(q_2, q_2)\}$.

$p_{q_1}$ and $p_{q_2}$ are RSTL, that the series $p = 2p_{q_1} - p_{q_2}$ computed by $\mathcal{A}$ takes only positive values. And since $\sum_t p(t) = 1$, $p$ is an RSTL. It admits the following linear representation: $(\mathbb{R}^2, \mu, \lambda)$ where $e_1 = (1, 0)$ and $e_2 = (0, 1)$ is a basis of $\mathbb{R}^2$, $\lambda(e_1) = 2, \lambda(e_2) = -1$, $\mu(a) = 2e_1/3 + 3e_2/4$, $\mu(f)(e_1, e_1) = e_1/3, \mu(f)(e_2, e_2) = e_2/4$ and $\mu(f)(e_i, e_j) = 0$ if $i \neq j$.

## 2.1 Canonical Linear Representation of Rational Tree Series

We now define the canonical representation of a rational tree series.
Let $c \in C(\mathcal{F})$. We define the linear mapping $\dot{c} : \mathbb{R}\langle\langle\mathcal{F}\rangle\rangle \to \mathbb{R}\langle\langle\mathcal{F}\rangle\rangle$ by

$$\dot{c}(r)(t) = r(c[t]) \ .$$

[1] These automata are also referred to as *multiplicity tree automata* in the literature.

Let $r \in \mathbb{R}\langle\langle \mathcal{F} \rangle\rangle$. Let us denote by $W_r$ the vector subspace of $\mathbb{R}\langle\langle \mathcal{F} \rangle\rangle$ spanned by $\{\dot{c}r | c \in C(\mathcal{F})\}$. It can be shown that $r$ is rational if and only if the dimension of $W_r$ is finite [1]. Let $W_r^*$ be the dual space of $W_r$, i.e. the set of all linear forms on $W_r$. For any $t \in T(\mathcal{F})$, let $\overline{t} \in W_r^*$ be defined by: $\forall s \in W_r, \ \ \overline{t}(s) = s(t)$. It can be shown that there exist trees $t_1, \ldots, t_n$ such that $\{\overline{t_1}, \ldots, \overline{t_n}\}$ forms a basis of $W_r^*$. Let us define the linear representation $(W_r^*, \mu, \lambda)$ as follows:

- for any $f \in \mathcal{F}_m$, define $\mu(f)(\overline{t_{i_1}}, \ldots, \overline{t_{i_m}}) = \overline{f(t_{i_1}, \ldots, t_{i_m})}$.
- $\lambda \in (W_r^*)^* = W_r$ by $\lambda(\overline{t}) = r(t)$.

**Theorem 1.** *[1] $(W_r^*, \nu, \tau)$ is a linear representation of $r$ which is called the* canonical linear representation *of $r$. It can be embedded in any linear representation of $r$; in particular, its dimension is minimal.*

*Example 2.* Consider the rational stochastic tree language $p$ defined in Example 1. It can easily be shown that $p_1(t) = \frac{2^{|t|_f+1}}{3^{2|t|_f+1}}$, $p_2(t) = \frac{3^{|t|_f+1}}{4^{2|t|_f+1}}$ and $p(t) = \frac{2^{5|t|_f+4} - 3^{3|t|_f+2}}{3^{2|t|_f+1} \times 4^{2|t|_f+1}}$. Thus, for any context $c$ and any tree $t$:

$$\overline{t}(\dot{c}p) = p(c[t]) = \frac{2^{5|t|_f+5|c|_f+4} - 3^{3|t|_f+3|c|_f+2}}{3^{2|t|_f+2|c|_f+1} \times 4^{2|t|_f+2|c|_f+1}}.$$

Since $p$ has a 2-dimensional linear representation, the dimension of $W_r^*$ is $\leq 2$. Let $c_0 = \$$ and $c_1 = f(a, \$)$, we have:

$$\overline{a}(\dot{c}_0 p) = \frac{7}{3 \times 2^2}, \overline{a}(\dot{c}_1 p) = \overline{f(a,a)}(c_0) = \frac{269}{3^3 \times 2^6}, \text{ and } \overline{f(a,a)}(\dot{c}_1 p) = \frac{9823}{3^5 \times 2^{10}}.$$

Since $\overline{a}(\dot{c}_0 p) \times \overline{f(a,a)}(\dot{c}_1 p) \neq \overline{a}(\dot{c}_1 p) \times \overline{f(a,a)}(\dot{c}_0 p)$, $\overline{a}$ and $\overline{f(a,a)}$ are linearly independent. Then, $\{\overline{a}, \overline{f(a,a)}\}$ is a basis of $W_r^*$. We define $\lambda$ and $\mu$ by:

$$\begin{gathered}\lambda(\overline{a}) = p(a) = \tfrac{7}{3 \times 2^2} \text{ and } \lambda(\overline{f(a,a)}) = p(f(a,a)) = \tfrac{269}{3^3 \times 2^6} \text{ and} \\ \mu(a) = \overline{a}, \ \mu(f)(\overline{a}, \overline{a}) = \overline{f(a,a)}, \\ \mu(f)(\overline{a}, \overline{f(a,a)}) = \mu(f)(\overline{f(a,a)}, \overline{a}) = \tfrac{-54}{2^4 \times 3^4}\overline{a} + \tfrac{59}{2^4 \times 3^2}\overline{f(a,a)}, \\ \mu(f)(\overline{f(a,a)}, \overline{f(a,a)}) = \tfrac{-3186}{2^8 \times 3^6}\overline{a} + \tfrac{2617}{2^8 \times 3^4}\overline{f(a,a)}.\end{gathered}$$

We can justify here why the canonical form of a stochastic language $p$ may not be relevant for generating trees according to $p$. Indeed, one can remark here that $\mu(f(a, f(a,a))) = \mu(f)(\overline{a}, \overline{f(a,a)}) = \frac{-54}{2^4 \times 3^4}\overline{a} + \frac{59}{2^4 \times 3^2}\overline{f(a,a)}$. Thus, if we consider the weights of trees according to $\overline{a}$, $f(a, f(a,a))$ has a negative weight and then $\overline{a}$ does not define by itself a stochastic language. As a consequence, the canonical form does not have a relevant structure if one aims at using it according to a generative model.

## 2.2 DEES

DEES is an inference algorithm which identifies any rational stochastic language in the limit with probability one (see [1]). Let us show how DEES works on the previous example. Let $S$ be a sample of trees independently drawn according to $p$ and let $p_S$ be the empirical distribution defined on $T(\mathcal{F})$: $p_S(t)$ is the frequency of $t$ in $S$. For any confidence parameter $\delta$, there exists $\epsilon > 0$ such that with probability at least $1-\delta$, $|p(t) - p_S(t)| \leq \epsilon$ for any tree $t$. Statistical tests, based on this property, are used to accept or reject hypotheses of the form: $\overline{t}$ is a linear combination of $\overline{t_1}, \ldots, \overline{t_n}$. Parameters $\epsilon$ and $\delta$ can be chosen, depending on the size of the sample $S$, such that with probability one, the correct hypothesis will always be chosen from some sample size.

In order to find the basis of the canonical representation, the algorithm first tests whether $\overline{a}$ and $\overline{f(a,a)}$ are linearly independent. With probability one, this will be detected from some step: $\overline{a}$ and $\overline{f(a,a)}$ are elements of the canonical basis. Then, the algorithm tests whether $\overline{f(a,f(a,a))}$ is a linear combination of $\overline{a}$ and $\overline{f(a,a)}$. As this is true, this will be detected with probability one from some step. Therefore, $f(a,f(a,a))$ will not be added to the basis. And so on. The algorithm terminates when it has checked that no more elements can be added to the basis.

It can be proved that with probability one, there exists an integer $N$ such that for any sample $S$ containing more than $N$ examples, a basis of $W_p^*$ will be identified from $S$. DEES will compute a linear representation $(W_p^*, \mu_S, \lambda_S)$, such that $\mu_S$ and $\lambda_S$ converge respectively to $\mu$ and $\lambda$ when $S$ tends to infinity.

Hence, DEES identifies in the limit the canonical linear representation of any rational tree stochastic language with probability one. However:

- Given the canonical linear representation of a stochastic language $p$ does not help to generate trees according to $p$.
- Moreover, the series output by DEES from some sample $S$ can be not a stochastic language. The possibility to transform it in a stochastic language is then an important issue.
- The series output by DEES converges to the target $p$ as the size of $S$ increases, but what is the rate of convergence?

We propose to address of all of these questions in the present paper.

## 3 Normalised Linear Representation for Rational Stochastic Tree Languages

### 3.1 Normalised Representation

Let $r$ be a rational stochastic tree language represented by a WTA $(Q, \mathcal{F}, \tau, \delta)$. Tree series $r_q$ associated with each state in $Q$ may not be stochastic tree languages. This is illustrated by Example 2. Trivially, the same remark can be made for the equivalent linear representation of tree series, considering the series associated with every basis vector. However, as stated by the following theorem, there exist equivalent representations, called normalised, where each tree series associated with basis vectors are indeed stochastic tree languages.

Let $\delta_{ij}$ be Kronecker symbols, $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

**Theorem 2.** *Let $p$ be an* RSTL *over $T(\mathcal{F})$ and let $(W_p^*, \mu, \lambda)$ be the canonical linear representation of $p$. Then, $W_p^*$ admits a basis $B = \{e_1, \dots, e_n\}$ such that each series $p_i$ defined by $(V, \mu, \lambda_i)$ where $\lambda_i(e_j) = \delta_{ij}$ is stochastic.*

*Proof.* Let $c_1, \dots, c_n \in C(\mathcal{F})$ such that $\{c_1^{-1}p, \dots, c_n^{-1}p\}$ is a basis of $W_p$. Let $\{\ell_1, \dots, \ell_n\}$ be a basis of $W_p^*$ such that $\ell_i(c_j^{-1}p) = \delta_{ij}$ for $1 \le i, j \le n$.
We show below that $\{\ell_1, \dots, \ell_n\}$ is a normalised basis of $W^*$. Let $\lambda_1, \dots, \lambda_n$ be the linear forms defined on $W^*$ by $\lambda_i(\ell_j) = \delta_{ij}$. Let us show that for any $t \in T(\mathcal{F})$ and any $1 \le i \le n$, $\lambda_i(\mu(t)) = c_i^{-1}p(t)$.

Let $\{\overline{t}_1, \dots, \overline{t}_n\}$ be a basis of $W_p^*$ and let $\overline{t_i} = \sum_{j=1}^{n} \gamma_i^j \ell_j$ for any $1 \le i \le n$. We have:

$$\overline{t_i}(c_j^{-1}p) = \sum_{k=1}^{n} \gamma_i^k \ell_k(c_j^{-1}p) = \sum_{k=1}^{n} \gamma_i^k \delta_{kj} = \gamma_i^j. \tag{1}$$

Let $t \in T(\mathcal{F})$ and $\overline{t} = \sum_{i=1}^{n} \beta_i \overline{t_i}$, then:

$$\overline{t} = \sum_{i=1}^{n} \left( \beta_i \sum_{j=1}^{n} \gamma_i^j \ell_j \right) = \sum_{j=1}^{n} \left( \sum_{i=1}^{n} \beta_i \gamma_i^j \right) \ell_j \quad . \tag{2}$$

Because $(W_p^*, \mu, \lambda)$ is the canonical representation of $p$, we have $\mu(t) = \overline{t}$ by definition. Hence,

$$\begin{aligned}
\lambda_j(\mu(t)) = \lambda_j(\overline{t}) &\overset{(2)}{=} \sum_{i=1}^{n} \beta_i \gamma_i^j \overset{(1)}{=} \sum_{i=1}^{n} \beta_i \overline{t_i}(c_j^{-1}p) \\
&= \overline{t}(c_j^{-1}p) \qquad\qquad \text{since } \overline{t} = \sum_{i=1}^{n} \beta_i \overline{t_i} \\
&= c_j^{-1}p(t).
\end{aligned}$$

Hence, $(W_p^*, \mu, \lambda_i)$ represents a stochastic language for $1 \le i \le n$. □

**Definition 2.** *Let $\mathcal{A} = (Q, \mathcal{F}, \tau, \delta)$ be a* WTA. *We say that $\mathcal{A}$ is in* normalised *form if and only if (i) $\sum_{q \in Q} \tau(q) = 1$, (ii) for any $q \in Q$, $\sum_{r \in \Delta_q} w(r) = 1$ and (iii) for any $q \in Q$ and any $f \in \mathcal{F}$, $\sum_{r \in \Delta_q} w(r) \in [0, 1]$. Moreover, we say that $\mathcal{A}$ is in* reduced *normalised form if the series $r_q$ are linearly independent.*

Any rational stochastic tree language can be represented by a normalised reduced WTA $\mathcal{A} = (Q, \mathcal{F}, \tau, \delta)$ such that each $r_q$ defines a stochastic language. Note also that any PTA is in normalised form (but not necessarily in reduced normalised form).

*Example 3.* Let us consider the rational stochastic tree language $p$ presented in the previous examples, we show how to compute a normalised WTA that computes it. Let $c_0 = \$$, $c_1 = f(\$, a)$ and let $s_0 = \alpha_0 \overline{a} + \beta_0 \overline{f(a, a)}$ and $s_1 = \alpha_1 \overline{a} +$

$\beta_1\overline{f(a,a)}$ where $s_i(c_j^{-1}p) = \delta_{ij}$. Remarking that $\sum_t \dot{c_0}p(t) = 1$ and $\sum_t \dot{c_1}p(t) = \sum_t p(f(t,a)) = 2\sum_t p_1(f(t,a)) - \sum_t p_2(f(t,a)) = 37/144$ one can check that $\alpha_0 = \frac{-9823}{300}, \alpha_1 = \frac{3228}{25}, \beta_0 = \frac{9953}{300}$ and $\beta_1 = \frac{-3108}{25}$.

Now, by expressing, $\overline{a}$ and $\overline{f(a,a)}$ in the basis $s_0, s_1$, we get the following set of rules:

$$\begin{array}{llllll}
s_0 & \xrightarrow{7/12} & a, & s_1 & \xrightarrow{269/444} & a, \\
s_0 & \xrightarrow{-269/50} & f(s_0,s_0), & s_1 & \xrightarrow{-3024/925} & f(s_0,s_0), \\
s_0 & \xrightarrow{259/50} & f(s_0,s_1), & s_1 & \xrightarrow{2664/925} & f(s_0,s_1), \\
s_0 & \xrightarrow{259/50} & f(s_1,s_0), & s_1 & \xrightarrow{2664/925} & f(s_1,s_0), \\
s_0 & \xrightarrow{-1369/300} & f(s_1,s_1), & s_1 & \xrightarrow{-23273/11100} & f(s_1,s_1).
\end{array}$$

Let $\lambda(s_0) = 1$ and $\lambda(s_1) = 0$. It is easy to verify that this representation is in normalised form.

### 3.2 A Generation Process

A generation process of trees can be done using normalized WTA as given in Algorithm 1. Each tree is built top-down. The process is different from the classical approach with PTA since instead of drawing a transition rule to apply at each step, we rather draw a symbol according to the distributions of the symbols defined by the rules.

**Comments** of the steps numbered by **(1)**, **(2)**, **(3)** and **(4)** in Algorithm 1:

**(1)** $\Delta_{gen}$ contains $n$ rules of the form $q_t \xrightarrow{w_i} c[q_1^i, \ldots, q_m^i]$ where $c$ is a linear context over $m$ variables and where $1 \le i \le n$.
**(2)** It can be proved that $\sum_{f\in\mathcal{F}} \alpha_{f,j}^c = 1$ for any $1 \le j \le m$.
**(3)** The numbers $\alpha_{f,j}^c$ define a probability distribution over $\mathcal{F}_j$.
**(4)** There exists a unique tree $t$ such that all the rules of $\Delta_{gen}$ are of the form $q_t \xrightarrow{w_i} t$; $t$ is the output of the algorithm.

## 4 Learning Rational Stochastic Tree Languages

We consider the question of learning a rational stochastic tree language (RSTL) $p$ from an i.i.d. sample of trees drawn according to $p$. An RSTL can be such that the average size of trees generated from $p$ is unbounded, i.e. $\sum_t p(t)|t| = \infty$. For example, this is the case for the RSTL defined by the PTA whose rules are: $\{q \xrightarrow{1/2} a, q \xrightarrow{1/2} f(q,q)\}$. To our knowledge, it is still unknown whether a PTA defines a RSTL and it is much better to deal with the stronger notion of *strongly consistent* stochastic language: A RSTL $p$ is strongly consistent if $\sum_t |t|p(t) < \infty$. Next section investigates some properties of strongly consistent RSTL.

```
Data    : An WTA $\mathcal{A} = (Q, \mathcal{F}, \tau, \delta)$ in normalised form
Result  : A tree $t \in T(\mathcal{F})$
begin
    Let $q_t$ be a new state ;
    Let $\Delta_{gen} = \{q_t \overset{\tau(q)}{\rightarrow} q | q \in Q\}$ (1);
    while the rhs of some rule of $\Delta_{gen}$ contains states do
        Let $m$ be the number of rules in $\Delta_{gen}$ and $n$ be the number of states
        in each the rules;
        for $1 \leq j \leq m$ do
            for any $f \in \mathcal{F}$, let $\alpha^c_{f,j} = \sum_{i=1}^{n} w_i \sum_{r \in \Delta_{q^i_j, f}} w(r)$ (2);
            draw randomly $f_j \in \mathcal{F}$ according to $\alpha^c_{f,j}$ (3);
        let $n_j$ be the rank of $f_j$;
        let $c' = c(f_1(\$^1_1, \ldots, \$^{n_1}_1), \ldots, f_m(\$^1_m, \ldots, \$^{n_m}_m))$ a linear context;
        in $\Delta_{gen}$, replace each rule $q_t \overset{w_i}{\rightarrow} c[q^i_1, \ldots, q^i_m]$ by the rules
        $q_t \overset{w_i w_{r_1} \ldots w_{r_m}}{\longrightarrow} c[f_1(q^1_{r_1}, \ldots, q^{n_1}_{r_1}), \ldots, f_m(q^1_{r_m}, \ldots, q^{n_m}_{r_m})]$ ;
        where $r_j : q_j \overset{w_{r_j}}{\rightarrow} f_j(q^1_{r_j}, \ldots, q^{n_1}_{r_j}) \in \Delta_{q^i_j, f_j}$, $1 \leq j \leq m$, $1 \leq i \leq n$;
    Outputs the tree of $\Delta_{gen}$ (4);
end
```

**Algorithm 1.** Drawing a tree according to a RSTL

## 4.1 Strongly Consistent Rational Stochastic Languages

Let $\mathcal{A} = (Q = \{q_1, \ldots, q_n\}, \mathcal{F}, \tau, \delta)$ be a WTA and let $A = (a_{ij})_{1 \leq i,j \leq n}$ be the matrix defined by $a_{ij} = \sum_{r \in \Delta_{q_i}} n_r(j) w(r)$ where $n_r(j)$ is the number of occurrences of $q_j$ in the rhs of $r$.

We denote by $p_i$ the rational series defined from state $q_i$ and we let $\gamma_i = \sum_{t \in T(\mathcal{F})} p_i(t)|t|$ ($\gamma_i$ may be undefined if the sum diverges), $\gamma = (\gamma_1, \ldots, \gamma_n)$ and $B = (1, \ldots, 1)^t$.

**Proposition 1.** *Let us suppose that for any index $i$,* $\sum_{t \in T(\mathcal{F})} p_i(t) = 1$ *and* $\sum_{r \in \Delta_{q_i}} w(r) = 1$. *Then* $\gamma = \sum_{n \geq 0} A^n B$.

*Proof.* The proof is detailed in [8].

The sum $\sum_{n \geq 0} A^n B$ converges iff $A^n B$ converges to 0, which can be decided within polynomial time.

*Example 4.* Consider the PTA defined by the rules $\{q \overset{1-\alpha}{\rightarrow} a, q \overset{\alpha}{\rightarrow} f(q, q)\}$ and $\tau(q) = 1$: $A = (2\alpha)$ and $A^n B$ converges iff $\alpha < 1/2$. The average size of trees generated from these PTA is $1/(1 - 2\alpha)$. When $\alpha = 1/3$ (resp. $1/4$), the PTA

computes the stochastic language $p_{q_1}$ (resp. $p_{q_2}$) as previously defined in example 1. Then, the average size of trees $\gamma_1$ (resp. $\gamma_2$) generated from $p_{q_1}$ (resp. $p_{q_2}$) is 3 (resp. 2). One can deduce the average size of the stochastic language $p = 2p_{q_1} - p_{q_2}$, $\gamma = 2 \times \gamma_1 - \gamma_2 = 4$.
Consider now the normalised form of $p$ as presented in example 3.
The matrix $A$ is $\begin{pmatrix} -2/5 & 37/30 \\ -144/185 & 47/30 \end{pmatrix}$.

It is easy to verify that $(I - A)$ is invertible and $(I-A)^{-1} = \begin{pmatrix} -17/5 & 37/5 \\ -864/185 & 42/5 \end{pmatrix}$.
Thus $(I-A)^{-1}B = \begin{pmatrix} 4 & 690/185 \end{pmatrix}$. Following Prop. 1, the average size $\gamma_0$ of trees generated by $c_0^{-1}p$ is 4 and the average size of trees generated by $c_0^{-1}p$ is 690/185. Since $p = c_0^{-1}p$ the average tree size of $p$ is 4.

We show below that when $\mathcal{A}$ is a reduced normalised representation of a strongly consistent rational stochastic language, the spectral radius[2] $\rho(A)$ of $A$ is $< 1$. We need the following lemma :

**Lemma 1.** *Let $p_1, \ldots, p_n$ be $n$ independent stochastic languages. Then $\Lambda = \{(\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n : \sum_{i=1}^{n} \alpha_i p_i$ is a stochastic language$\}$ is a compact convex subset of $\mathbb{R}^n$.*

*Proof.* See [9] for a similar proof in the case of words.

**Proposition 2.** *Let $\mathcal{A} = (Q = \{q_1, \ldots, q_n\}, \mathcal{F}, \tau, \delta)$, a reduced normalised representation of a strongly consistent* RSTL *$p$ such that each $p_{q_i}$ is a stochastic language and let $A = (a_{ij})_{1 \le i,j \le n}$ be the matrix defined as previously. Then the spectral radius of $A$ satisfies $\rho(A) < 1$.*

*Proof.* The proof is detailed in [8].

*Example 5.* The matrix $A$ of Example 4 admits two eigenvalues: $\frac{1}{2}$ and $\frac{2}{3}$, then $\rho(A) = \frac{2}{3} < 1$.

## 4.2 Effective Normalisation

Let $p$ be a strongly consistent RSTL and let $B = \{\overline{t_1}, \ldots, \overline{t_n}\}$ be the smallest (for the order $\le$ on $T(\mathcal{F})$) basis of the canonical linear representation $(W_p^*, \mu, \lambda)$ of $p$. The main result in [1] proves that with probability one, there exists a sample size from which DEES outputs a linear representation $(W_p^*, \mu_S, \lambda_S)$ whose basis is $B$ and such that $\mu_S$ and $\lambda_S$ are arbitrarily close to $\mu$ and $\lambda$.

Theorem 2 states that there exists a normalised WTA $\mathcal{A}_S$ given its canonical linear representation $(W_p^*, \mu, \lambda)$. In this section we explain how to effectively compute $\mathcal{A}_S$. Choosing a basis written as $\{\dot{c}_1 p, \ldots, \dot{c}_n p\}$ is easily done by recursively enumerating every context, the main technical key point relies in the ability to compute the sums $\sum_{t \in T(\mathcal{F})} p(c_i[t])$ for a given rational series.

[2] The spectral radius of a matrix is the maximum of the norms of its complex eigenvalues.

Let $s$ be the vector defined by $s = \sum_{t \in T(\mathcal{F})} \mu(t) = \sum_{t \in T(\mathcal{F})} \overline{t}$. The $i$th component of $s$ is $\sum_{t \in T(\mathcal{F})} p_i(t) = \sum_{t \in T(\mathcal{F})} p(c_i[t])$. Moreover, $s$ is a solution of the polynomial system: $v = F(v)$ where $F(v) = \sum_m \sum_{f \in \mathcal{F}_m} \mu(f)(v, \ldots, v)$. This system is not analytically soluble in general. As a consequence, we approximate $s$ using with a direct propagative method.

Let $E$ and $E_k$ be the endomorphisms defined by:

$$E(v) = \sum_m \sum_{l=1}^{m} \sum_{f \in \mathcal{F}_m} \mu(f)(\underbrace{s, \ldots, s}_{l-1}, v, \underbrace{s, \ldots, s}_{m-l})$$

$$E_k(v) = \sum_m \sum_{l=1}^{m} \sum_{f \in \mathcal{F}_m} \mu(f)(\underbrace{s_k, \ldots, s_k}_{l-1}, v, \underbrace{s, \ldots, s}_{m-l}) .$$

A propagative method is proposed by Stolcke[10] in the case of probabilistic context-free languages. Let $T^{<k}(\mathcal{F})$ be the set of trees of height lower than $k$. The idea is to recursively compute the sequence $s_k = \sum_{t \in T^{<k}(\mathcal{F})} \overline{t}$ using the recursion: $s_0 = 0$ and $s_{k+1} = F(s_k)$. Obviously, $(s_k)$ converges towards $s$. Let us study the convergence rate.

By applying the multi-linearity of $\mu(f)$, $s - s_{k+1}$ can be decomposed in $s - s_{k+1} = F(s) - F(s_k) = E_k(s - s_k)$. Taking into account that for every tree $t$, the $i$th component of $\overline{t}$ is $p(c_i[t]) \geq 0$, it is easily shown that for every $k$:

$$\|s - s_{k+1}\| = \|\prod_{q=0}^{k} E_q(s - s_0)\| \leq \|E^k\| \|(s - s_0)\| \ .$$

By Gerland's formula, we have $\|E^k\| \sim \rho(E)^k$ and thus:

$$\|s - s_k\| = O(\rho(E)^k \|s - s_0\|) \ .$$

Let $A$ be the matrix of $E$ in the basis $\{c_1^{-1}p, \ldots, c_n^{-1}p\}$. It can be proved that $A$ is the same matrix as defined in Section 4.1. Thanks to Proposition 2 and because we made the assumption the series is strongly consistent, we know that $\rho(E) = \rho(A) < 1$.

When tested on the previous example, the propagative method achieved precision of $10^6$ in approximately 30 iterations. In near future, we intend to study the use of Newton's method, which could at least theoretically achieve faster convergence.

## 4.3 Learning a Strongly Consistent Rational Stochastic Language: The Road Map

The normalised WTA $\mathcal{A}_S$ obtained at the end of the previous section computes an RSTL $p_S$ such that the spectral radius $\rho_S$ of the matrix $A_S$ associated with $\mathcal{A}_S$ satisfies $\rho_S < 1$ which is a strong property. We have still some results to prove in order to complete the learning process. We present them below as conjectures.

**Conjecture 1:** It is possible to modify Algorithm 1 in order to be used to generate trees from a normalised WTA (even when it does not define a stochastic language). The modified algorithm stops (and outputs a tree) with probability one, as soon as $S$ is sufficiently large. Hence, it defines a stochastic language $\hat{p}$.

**Conjecture 2:** with probability one, $\sum_t |p(t) - \hat{p}(t)| \cdot |t|$ converges to 0 with the size of $S$.

These two conjectures generalize results proved in the word case. Note that the convergence type described in Conjecture 2 is stronger than $L_1$-convergence.

## 5 Unranked Trees

In this section we consider trees where the rank constraint has been dropped: Every symbol in unranked trees may have from 0 to an unbounded but finite number of (ordered) children. Unranked trees are the common abstract representation of semi-structured data like XML.

Let $\Sigma$ be a finite set of symbols. The set $T(\Sigma)$ of unranked trees is the smallest set such that $\Sigma \subseteq T(\Sigma)$, and $f(t_1, \ldots, t_m) \in T(\Sigma)$ provided $f \in \Sigma$ and $t_1, \ldots, t_m \in T(\Sigma)$. An algebraic definition of unranked trees can be given by means of the extension operator @ ([4]). Basically, @ adds a new child at the end of the list of children of an unranked tree: $f \,@\, t = f(t)$, $f(t_1, \ldots, t_{n-1}) \,@\, t_n = f(t_1, \ldots, t_n)$ .

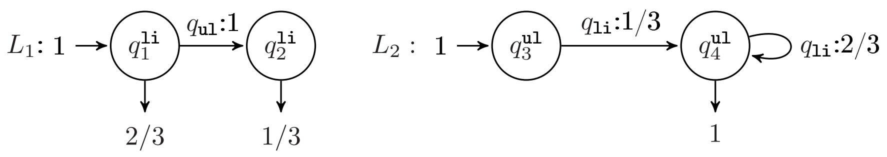
The extension operator provides a unique recursive definition of any unranked tree. It can be syntactically represented by a binary (ranked) tree over $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_2$ where $\mathcal{F}_0 = \Sigma$ and $\mathcal{F}_2 = \{@\}$. Let us now define the mapping $\mathsf{ext}$ from $T(\Sigma)$ to $T(\mathcal{F})$ by $\mathsf{ext}(f) = f$ and $\mathsf{ext}(f(t_1, \ldots, t_n)) = @(\mathsf{ext}(f(t_1, \ldots, t_{n-1})), \mathsf{ext}(t_n))$. One can show that the mapping $\mathsf{ext}$ is a bijection. Hedge automata [11] directly act on unranked trees in $T(\Sigma)$. Briefly, hedge automata rules are of the form $f(L) \to q$ where $L$ is a word language on the alphabet of states. It has be shown that hedge automata and ordinary tree automata on $T(\mathcal{F})$ define the same class of recognizable languages [12]. Extension from hedge automata to weighted hedge automata (there referred to as unranked WTA) is proposed in [13]. In unranked WTA rules are of the form $f(L) \xrightarrow{w} q$ where $L$ is a weighted word language on the alphabet of states.

Thanks to the $\mathsf{ext}$ mapping, each result presented in this paper can be interpreted in the case of unranked trees. Tree series on $T(\Sigma)$ are simply defined via tree series on $T(\mathcal{F})$. This mapping also suggests a notion of rational unranked tree series and stochastic languages.

**Proposition 3.** *The class of rational unranked tree series represented via the mapping* $\mathsf{ext}$ *coincide with the class of unranked tree series defined by unranked* WTA.

More precisely, let be an unranked WTA which represents a rational unranked tree series $r^u$. One can build in linear time a (ranked) WTA which represents a rational tree series $r^r$ such that $\forall t \in T(\Sigma)\ r^u(t) = r^r(\mathsf{ext}(t))$. The converse is also true but to compute the corresponding unranked WTA, one needs to normalise rules following the method given in Section 4.2.

The following example illustrates how one can build a weighted automaton for unranked trees. Let us consider trees that represent nested lists built with the commonly used symbols `ul` and `li`. Let us consider first a stochastic hedge automaton with two states $q_{\texttt{ul}}$ and $q_{\texttt{li}}$. Final weights are given by $F(q_{\texttt{ul}}) = 1$ and $F(q_{\texttt{li}}) = 0$. Rules are $\texttt{li}(L_1) \xrightarrow{1} q_{\texttt{li}}$ and $\texttt{ul}(L_2) \xrightarrow{1} q_{\texttt{ul}}$ where



The weight of a tree $\texttt{ul}(\texttt{li}, \texttt{li}(\texttt{ul}(\texttt{li})))$ is $2^3/3^6$.

The corresponding automaton on the expression with the @ operator has 4 states $\{q_1^{\texttt{li}}, q_2^{\texttt{li}}, q_3^{\texttt{ul}}, q_4^{\texttt{ul}}\}$, $\tau(q_4^{\texttt{ul}}) = 1$ and the set of rules:

$$\left\{ \begin{array}{ccc} \texttt{li} \xrightarrow{1} q_1^{\texttt{li}} \quad , \quad \texttt{ul} \xrightarrow{1} q_3^{\texttt{ul}} & , & @(q_1^{\texttt{li}}, q_4^{\texttt{ul}}) \xrightarrow{w_1} q_2^{\texttt{li}}, \\ @(q_3^{\texttt{ul}}, q_1^{\texttt{li}}) \xrightarrow{w_2} q_4^{\texttt{ul}} & , & @(q_3^{\texttt{ul}}, q_2^{\texttt{li}}) \xrightarrow{w_3} q_4^{\texttt{ul}}, \\ @(q_4^{\texttt{ul}}, q_1^{\texttt{li}}) \xrightarrow{w_4} q_4^{\texttt{ul}} & , & @(q_4^{\texttt{ul}}, q_2^{\texttt{li}}) \xrightarrow{w_5} q_4^{\texttt{ul}} \end{array} \right\}$$

The weight $w_2$ is the weight of adjoining a subtree in state $q_1^{\texttt{li}}$ to a tree in state $q_3^{\texttt{ul}}$. The results gives a tree in state $q_4^{\texttt{ul}}$. It corresponds to the following computation in the hedge automaton: exit from $L_1$ with state $q_1^{\texttt{li}}$, then apply the rule $\texttt{li}(L_1) \xrightarrow{1} q_{\texttt{li}}$ and finally follow the transition from $q_3^{\texttt{ul}}$ to $q_4^{\texttt{ul}}$ in $L_2$. Hence $w_2 = 2/3 \times 1 \times 1/3$. Similarly $w_3 = 1/3 \times 1 \times 1/3$, $w_4 = 2/3 \times 1 \times 2/3$, $w_5 = 1/3 \times 1 \times 2/3$ and $w_1 = 1 \times 1 \times 1$. The binary tree associated with $\texttt{ul}(\texttt{li}, \texttt{li}(\texttt{ul}(\texttt{li})))$ is $@(@(\texttt{ul}, \texttt{li}), @(\texttt{li}, @(\texttt{ul}, \texttt{li})))$. One can verify that its weight is also $2^3/3^6$.

Hence, to learn rational unranked tree series, one can simply proceed in the following way: apply $\mathsf{ext}$ to input trees and then apply DEES. Eventually, a representation of an unranked WTA where weights are estimated can possibly be returned.

## 6 Conclusion

In this paper, we studied the problem of learning a rational stochastic tree language $p$ from an *i.i.d.* sample of trees drawn from $p$. An inference algorithm, DEES, was previously proposed for this problem. Using this algorithm leads to two main drawbacks: It often outputs linear representations that do not define stochastic languages and these representations can not be directly used to generate trees from the underlying distribution. We addressed this problem by showing that any rational stochastic tree language admits a normalised reduced representation that can be used as a generative model. We have studied the notion of strongly consistent rational stochastic languages which corresponds to the fact that the average size of trees generated from a RSTL $p$ is bounded. We showed the relationship between this notion and the normalised reduced representation of a RSTL. We finally justified that the methods presented in this paper can be directly applied to unranked trees.

The next step of this work is to prove the conjectures that was presented for learning strongly consistent rational stochastic languages: First, a probability distribution $\hat{p}$ can be extracted in order to generate trees from a normalised WTA. Second, that $\sum_t |p(t) - \hat{p}(t)| \cdot |t|$ convergences to zero with the size of the learning sample. Note here that this condition is stronger than the $L_1$-convergence.

## References

1. Denis, F., Habrard, A.: Learning rational stochastic tree languages. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 242–256. Springer, Heidelberg (2007)
2. Booth, T., Thompson, R.: Applying probabilistic measures to abstract languages. IEEE Transactions on Computers 22(5), 442–450 (1973)
3. Wetherell, C.S.: Probabilistic languages: A review and some open questions. ACM Comput. Surv. 12(4), 361–379 (1980)
4. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007) (release October 12, 2007), http://tata.gforge.inria.fr/
5. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoretical Computer Science 18, 115–148 (1982)
6. Ésik, Z., Kuich, W.: Formal tree series. Journal of Automata, Languages and Combinatorics 8(2), 219–285 (2003)
7. Denis, F., Esposito, Y.: Rational stochastic languages. Technical report, LIF - Université de Provence (2006), http://hal.ccsd.cnrs.fr/ccsd-00019728
8. Denis, F., Gilbert, E., Habrard, A., Ouardi, F., Tommasi, M.: Relevant representations for the inference of rational stochastic tree languages. Technical report, LIF, LIFL, and INRIA (2008), http://hal.archives-ouvertes.fr/hal-00293511/en/
9. Denis, F., Esposito, Y., Habrard, A.: Learning rational stochastic languages. In: Lugosi, G., Simon, H.U. (eds.) Learning theory. LNCS, pp. 274–288. Springer, Heidelberg (2006)
10. Stolcke, A.: An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computional Linguistics 21(2), 165–201 (1995)
11. Brüggemann-Klein, A., Murata, M., Wood, D.: Regular tree and regular hedge languages over unranked alphabets. Technical report, Hong Kong University Theoretical Computer Science Center, Version 1 (2001)
12. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 105–118. Springer, Heidelberg (2004)
13. Droste, M., Vogler, H.: Weighted logics for XML (manuscript, 2007), http://www.orchid.inf.tu-dresden.de/gdp/monographs/r20.ps

# Learning Commutative Regular Languages[*]

Antonio Cano Gómez and Gloria I. Álvarez

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia. Valencia (Spain)
Departamento de Ciencias e Ingeniería de la Computación,
Pontificia Universidad Javeriana Cali, calle 18 No. 118-250 Cali, Colombia
{acano,galvarez}@dsic.upv.es

**Abstract.** In this article we study the inference of commutative regular languages. We first show that commutative regular languages are not inferable from positive samples, and then we study the possible improvement of inference from positive and negative samples. We propose a polynomial algorithm to infer commutative regular languages from positive and negative samples, and we show, from experimental results, that far from being a theoretical algorithm, it produces very high recognition rates in comparison with classical inference algorithms.

## 1 Introduction

Regular languages are not inferable from positive samples is a well known result from Angluin [2]. This means that only inference from positive and negative samples is allowed. Nevertheless, the most useful algorithms for learning languages from positive and negative samples are not enough efficient to be applied in practice when looking at the recognition rate, mainly *RPNI* [11,10] and *red-blue* [4,9]. In [1] has been introduced the idea of trying to learn some subclasses of regular languages from positive and negative samples with the aim of improving the efficiency of the learning process, either if the considered subclass is not inferable from positive samples, the inference of those classes from positive and negative samples can be improved. In this article we study the class of commutative regular languages proposing a new algorithm called CRPNI and showing experimentally that CRPNI outperforms significantly recognition rates obtained with classical inference algorithms like red-blue and RPNI. Despite the algorithm given in this article follows the ideas of [1], actually both algorithms are very different in concepts and implementation.

In Section 2 we give the most important definitions about words and languages. We define in this sections the concepts of deterministic finite automata and Moore machine, that would be slightly modified for the commutative case in section 3.

Section 3 is devoted to the inference of regular commutative languages. First, we prove that commutative language are not inferable from positive samples,

[*] Work partially supported by Ministerio de Educación y Ciencia under project TIN2007-60760.

and then we give an algorithm for the inference of commutative regular languages from positive and negative samples. For this algorithm, first, we modify the definition of deterministic finite automata and Moore machine defining commutative deterministic finite automata and commutative Moore machine, these models consider the fact that languages with which we are working are commutative. We also prove that any commutative language can be recognised by a commutative deterministic finite automata. Nevertheless, the minimal deterministic finite automata is not equivalent to the minimal commutative deterministic finite automata. Finally we give our algorithm *CRPNI*, that is inspired in the RPNI [11] algorithm and we show its convergence in the limit.

In section 4 we analyse experimental results obtained from our implementation of the algorithm. We study two cases $|\Sigma| = 2$ and $|\Sigma| = 3$. In both cases we can see a great improvement of the CRPNI recognition rates with respect to two of the most used algorithms, namely redblue and RPNI.

## 2 Preliminaries

In this section we give the most important concepts used in this article. For further details about these concepts, the reader is referred to [8], [11] and [12]. In this paper we will use Moore machines in order to define the algorithm but also other machines like unbiased finite state automata [3] or the finite state classifiers [6] could be used. It could be interesting whether an implementation with those machines could improve the inference process.

### 2.1 Formal Languages and Automata

Throughout this paper if $\Sigma$ denotes a finite alphabet, $\Sigma^*$ denotes the free monoid generated by $\Sigma$ with the concatenation as the internal law and $\lambda$ as the neutral element. A *language* $L$ over $\Sigma$ is a subset of $\Sigma^*$ and the elements of $\Sigma^*$ are called *words*. Given $x \in \Sigma^*$, if $x = uv$ with $u, v \in \Sigma^*$, then $u$ (resp. $v$) is called *prefix* (resp. *suffix*) of $x$. Given a language $L \in \Sigma^*$ we denote by $Pre(L)$ ($Suf(L)$) the set of prefixes (resp. suffixes) of all words in the language $L$. Given a total order $<$ on $\Sigma$ we can define an order $<_{lex}$ on $\Sigma^*$ by setting for two words $u, v \in \Sigma^*$, $u <_{lex} v$ if $|u| < |v|$ or $|u| = |v|$ and there exists $x, y_1, y_2 \in \Sigma^*$ such that $u = xay_1$ and $xby_2$ with $a < b$. Given a word $w$ on an alphabet $\Sigma$ and one letter $a \in \Sigma$, we denote by $\pi_a(w)$ the projection of $w$ in $a$, for instance, if $w = abbcaa$, then $\pi_a(w) = aaa$. Given a language $L$ we define the *syntactic left congruence* as the left-congruence on $\Sigma^*$ defined as $u \sim_L v$ if and only if for any $x \in \Sigma^*, ux \in L \Leftrightarrow vx \in L$.

A *deterministic finite automaton* (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where $Q$ is a finite set of states, $\Sigma$ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ if the set of final states and $\delta$ is a partial function that maps $Q \times \Sigma$ in $Q$, which can be easily extended to words. A word $x$ is accepted by an automaton $\mathcal{A}$ if $\delta(q_0, x) \in F$. The set of words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Given a language $L$, the *minimal deterministic finite automaton* of $L$ is the automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{[x]_{\sim_L} \mid x \in \Sigma^*\}$, $\delta([x]_{\sim_L}, y) = [xy]_{\sim_L}$ for any $x, y \in \Sigma^*$, $q_0 = [\lambda]_{\sim_L}$, $F = \{[x]_{\sim_L} \mid x \in L\}$.

A Moore machine is a 6-tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ where $\Sigma$ (resp. $\Gamma$) is an input (resp. output) alphabet, $\delta$ is a partial function that maps $Q \times \Sigma$ in $Q$, and $\Phi$ is a function that maps $Q$ in $\Gamma$ called *output function*. The behaviour of $\mathcal{M}$ is given by the function $t_{\mathcal{M}} : \Sigma^* \to \Gamma$ defined as $t_{\mathcal{M}}(x) = \Phi(\delta(q_0, x))$ for every $x \in \Sigma^*$ such that $\delta(q_0, x)$ is defined.

Given a Moore machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ with $\Gamma = \{0, 1, \uparrow\}$ we can associate an automaton $\mathcal{A}_{\mathcal{M}} = (Q, \Sigma, \delta, q_0, F)$ where $F = \{q \in Q \mid \Phi(q) = 1\}$.

Given two finite sets of words $D_+$ and $D_-$, we define the $(D_+, D_-)$-*prefix tree machine* ($PTM(D_+, D_-)$) as the Moore machine having $\Gamma = \{0, 1, \uparrow\}$, $Q = Pre(D_+ \cup D_-)$, $q_0 = \lambda$ and $\delta(u, a) = ua$ if $u, ua \in Q$ and $a \in \Sigma$. For every state $u$, the value of the output function $\Phi$ associated to $u$ is 1, 0 and $\uparrow$ (undefined) depending whether $u$ belongs to $D_+$, to $D_-$ or to the complementary set of $D_+ \cup D_-$.

A Moore Machine $\mathcal{M} = (Q, \Sigma, \Gamma, \cdot, q_0, \Phi)$ is consistent with $(D_+, D_-)$ if for every $x \in D_+$ we have $\Phi(\delta(q_0, x)) = 1$ and for every $x \in D_-$ we have $\Phi(\delta(q_0, x)) = 0$.

## 2.2 Commutative Languages

Given two words $u$ and $v$ we say the they are commutatively equivalent if $u = a_1 a_2 \cdots a_n$ with $a_i \in \Sigma$ for $1 \leq i \leq$, and there exist a permutation $\sigma$ on $\{1, 2, \ldots, n\}$ such that $a_{\sigma(1)} a_{\sigma(2)} \cdots a_{\sigma(n)} = v$. We denote it by $u \sim_{com} v$. For instance, $abca \sim_{com} cbaa$.

Given an alphabet $\Sigma$, a language $L$ is commutative if and only if it is the union of some $\sim_{com}$-classes.

It seems to be some relation between *planar languages* and commutative languages [5] and their inference. An interesting work would be to compare the inference algorithm described in [5] and the CRPNI described here.

In this article we are interested in commutative regular languages. In order to describe the expressively of commutative regular languages we recall here a result from [12].

**Proposition 1.** (Pin) *For every alphabet $\Sigma$, the class of commutative languages of $\Sigma$ is the boolean algebra generated by the languages of the form $K(a, r) = u \in \Sigma^* \mid |u|_a = r$, where $r > 0$ and $a \in \Sigma$, or $L(a, k, p^n) = \{u \in \Sigma^* \mid |u|_a \equiv k \mod p^n\}$, where $0 \leq k < p^n$, $p$ is prime, $n > 0$ and $a \in \Sigma$*

Note that Proposition 1 show that the idea of inferring a language for each letter and trying to infer the whole language from those using boolean operation is not possible. For instance, $L = \{x \in \Sigma^* \mid |x|_a \equiv |x|_b \mod 2\}$ is a commutative language that can not be inferred in that way.

More equivalent definitions and proofs about commutative regular languages can be found in [12] and [13].

We now introduce the concept of commutative deterministic finite automata. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet. We define a *commutative deterministic finite automaton* (CDFA) on $\Sigma$ as $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where $Q = Q_{a_1} \times Q_{a_2} \times \cdots \times Q_{a_n}$, $q_0 \in Q$, $F \subseteq Q$, and $\delta((q1, \ldots, q_i, \ldots, q_n), a_i) = (q1, \ldots, \delta_{a_i}(q_i, a_i), \ldots, q_n)$ where $\delta_{a_i}$ is a function from $Q_{a_i}$ onto $Q_{a_i}$ for $1 \leq i \leq n$.

Let $\Sigma = \{a_1, a_3, \ldots, a_n\}$ be an alphabet and let $L$ be a commutative language on $\Sigma$. We define define the *minimal commutative automaton* of $L$ as $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where $Q = Q_{a_1} \times \ldots \times Q_{a_n}$ being $Q_{a_i} = \bigcup_{m \leq 0} [a_i^m]_{\sim L}$ (note that this union is finite) for $1 \leq i \leq n$, $q_0 = ([\lambda]_{\sim_L}, \ldots, [\lambda]_{\sim_L})$, $F = \{(\pi_{a_1}(x), \ldots, \pi_{a_n}(x)) \mid x \in L\}$ and $\delta_{a_i}([a_i^m]_{\sim_L}, a_i) = [a_i^{m+1}]_{\sim_L}$ for $1 \leq i \leq n$.

**Proposition 2.** *For any regular commutative language $L$, the minimal commutative deterministic automaton is well defined and accepts $L$.*

*Proof.* Let $\Sigma = \{a_1, \ldots, a_n\}$ and let $L$ be a commutative regular language. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be the minimal commutative finite automaton $L$. First, let $x \in L$, since $L$ is commutative we have that $\pi_{a_1}(x)\ \pi_{a_2}(x) \cdots \pi_{a_n}(x) \in L$, then we have, by the definition of $\delta$ that $\delta(q_0, x) = \delta(([\lambda]_{\sim_L}, [\lambda]_{\sim_L}, \ldots, [\lambda]_{\sim_L}), x) = ([\pi_{a_1}(x)]_{\sim_L}, [\pi_{a_2}(x)]_{\sim_L}, \cdots, [\pi_{a_n}(x)]_{\sim_L})$ and then by definition $([\pi_{a_1}(x)]_{\sim_L}, [\pi_{a_2}(x)]_{\sim_L}, \cdots, [\pi_{a_n}(x)]_{\sim_L})$ belongs to $F$, that is, $x$ is accepted by $A$. Now, let $x$ be a word accepted by $A$, then by definition we have that $\delta(q_0, x) = \delta(([\lambda]_{\sim_L}, [\lambda]_{\sim_L}, \ldots, [\lambda]_{\sim_L}), x) = ([\pi_{a_1}(x)]_{\sim_L}, [\pi_{a_2}(x)]_{\sim_L}, \cdots, [\pi_{a_n}(x)]_{\sim_L}) \in F$. Now, by the definition of $F$ we have necessarily $\pi_{a_1}(x)\pi_{a_2}(x) \cdots \pi_{a_n}(x) \in L$ and since $L$ is commutative we also have $x \in L$.

Note that this proposition does not assure that the minimal automaton of a given language $L$ is a commutative deterministic finite automata. For instance, for the language $L = \{x \in \Sigma^* \mid |x|_a = 0 \text{ or } |x|_b > 0\}$ we have that the minimal DFA and the minimal CDFA are not the same as shown in Figure 1.

The relationship between the size of the minimal DFA and the size of the minimal CDFA for a given commutative regular language is a interesting question

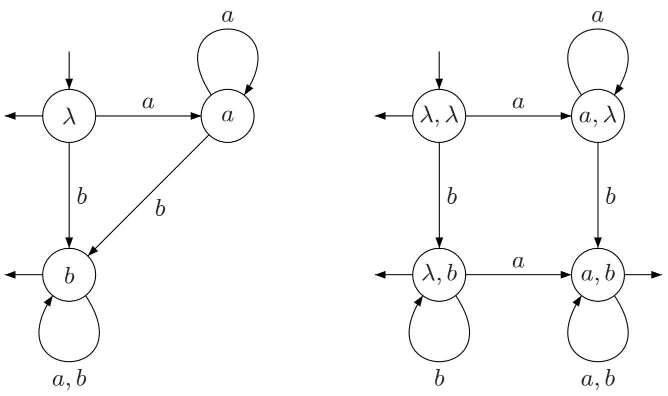


**Fig. 1.** The minimal DFA (on the left) and minimal CDFA (on the right) of the language $\{x \in \Sigma^* \mid |x|_a = 0 \text{ or } |x|_b > 0\}$

that would require an special study. The size of the alphabet seems to be a factor that would increase considerably the difference between both sizes.

We can also adapt the definition of Moore machine to the commutative case.

A *commutative Moore machine* is a 6-tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ where $\Sigma$ (resp. $\Gamma$) is an input (resp. output) alphabet, $Q = Q_{a_1} \times Q_{a_2} \times \cdots \times Q_{a_n}$, (where all $Q_{a_i}$ for $1 \leq i \leq n$ are finite sets of states ), $q_0 \in Q$, $\delta((q1, \ldots, q_i, \ldots, q_n), a_i) = (q1, \ldots, \delta_{a_i}(q_i, a_i), \ldots, q_n)$ where $\delta_{a_i}$ is a function from $Q_{a_i}$ onto $Q_{a_i}$ for any $1 \leq i \leq n$ and $\Phi$ is a function that maps $Q$ in $\Gamma$ called *output function.* The behaviour of $\mathcal{M}$ is given by a partial function $t_{\mathcal{M}} : \Sigma^* \to \Gamma$ defined as $t_{\mathcal{M}}(x) =$ for every $x \in \Sigma^*$ such that $\Phi(\delta(q_0, x))$ is defined.

Given a commutative Moore machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ with $\Gamma = \{0, 1, \uparrow\}$ we can associate a commutative automaton $\mathcal{A}_{\mathcal{M}} = (Q, \Sigma, \delta, q_0, F)$ where $F = \{q \in Q \mid \Phi(q) = 1\}$.

Given two finite sets of words $D_+$ and $D_-$ with $D_+ \cap D_- = \emptyset$, we define the $(D_+, D_-)$-*commutative prefix acceptor machine* $(CPAM(D_+, D_-))$ as the commutative Moore machine having $\Gamma = \{0, 1, \uparrow\}$, $Q = (\pi_{a_1}(x), \pi_{a_2}(x), \ldots, \pi_{a_n}(x))$ with $x \in Pre(D_+ \cup D_-)$, $q_0 = (\lambda, \ldots, \lambda)$ and where $\delta((u_1, \ldots, u_i, \ldots, u_n), a_i) = (u_1, \ldots, u_i a_i, \ldots, u_n)$ if $(u_1, \ldots, u_i, \ \ldots \ , u_n)$ , $(u_1, \ \ldots \ , u_i a_i, \ \ldots \ , u_n) \in Q$ and $a \in \Sigma$. For every state $(u_1, u_2, \ldots, u_n)$, the value of the output function $\Phi$ associated to $(u_1, u_2, \ldots, u_n)$ is 1, 0 depending whether there exist $x$ such that $\pi_{a_1}(x) = u_1, \pi_{a_2}(x) = u_2$ , ... , $\pi_{a_n}(x) = u_n$ and belonging to $D_+$, or to $D_-$. In other case the value is $\uparrow$ (undefined).

A Moore machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ is consistent with $(D_+, D_-)$ if for every $x \in D_+$ and any $y \sim_{com} x$ we have $\Phi(\delta(q_0, y)) = 1$ and for every $x \in D_-$ and any $y \sim_{com} x$ we have $\Phi(\delta(q_0, y)) = 0$.

## 3 Inference of Commutative Regular Languages

In this section we study the inference of commutative regular languages from positive samples and from positive and negative samples.

Firstly, we show that commutative regular languages are not inferable from positive samples, and so, no algorithm could be given for the inference.

**Proposition 3.** *Commutative regular languages are not inferable from positive samples.*

*Proof.* In order to show that the family of regular commutative languages is not inferable from positive samples, we take the family of languages $F = \bigcup_{n \geq 0}\{a^i \mid i \leq n\} \cup a^*$ where all these languages are commutative, and then $F$ also is. Now $F$ is superfinite, then by a standard result of [7] we conclude that the class of commutative languages is not inferable from positive samples.

Now, since any regular language is inferable from positive and negative samples, we describe here a new algorithm which allows us to infer faster commutative regular languages from positive and negatives samples.

## 3.1 Description of the Algorithm *CRPNI*

In this section we describe an inference algorithm from negative and positive samples (see Algorithm 1) for commutative regular languages called *commutative regular positive negative inference (CRPNI)*.

The definition of this algorithm is quite close to the description of the RPNI algorithm, being the main difference the sort of states we are working with. The main differences between them resides in the definition of $CPAM(D_+, D_-)$ and $PTM(D_+, D_-)$ and how the merge operation works as consequence of the definition of the states. For instance, in algorithm 1 any couple of states $(p_a, q_a)$ belong necessarily $\Sigma_a$ for some $a \in \Sigma$.

**Algorithm 1.** *CRPNI*$(D_+, D_-)$

```
1.  $M = CPAM(D_+ \cup D_-)$
2.  $\Sigma = alphabet(D_+ \cup D_-)$
3.  $listcomp = generateComparisonOrder(M, \Sigma)$
4.  while $listcomp \neq \emptyset$ do
5.    $(p_a, q_a) = first(listComp)$ (with $p_a, q_a \in Q_a$ for some $a \in \Sigma$)
6.    $listcomp = listcomp \backslash (p_a, q_a)$
7.    $push(queue, (p_a, q_a))$
8.    $M' = M;$
9.    $finish = false;$
10.   while $\neg emptyset(queue)$ and $\neg finish$ do
11.     $(p_a, q_a) = pop(queue)$
12.     if $\neg merge(M, p_a, q_a)$ then
13.       $M = M'$
14.       $finish = true$
15.     else
16.       $queue = detectNoDeterminism(queue, M)$
17.     end if
18.   end while
19. end while
20. Return $M$
```

The algorithm 1 considers all couples of elements in the order given by subroutine *generateComparisonOrder*, that is, considering the pairs in ascending order on the numerical consecutive and one different symbol each time; except possibly at the end if just one symbol remains with pairs unprocessed. Once a couple is chosen, the algorithm saves a copy of the current machine $M$ and tries to merge these elements into the states of the machine. Subroutine $merge(M, p_a, q_a)$ (with $p_a, q_a \in Q_a$ for some $a \in \Sigma$) merges $p_a, q_a$ in $Q_a$ and consequently any two states $(p_1, \dots, p_a, \dots, p_n)$ and $(q_1, \dots, q_a, \dots, q_n)$ in $Q$. If the merge is successful, $M$ changes and subroutine *detectNoDeterminism* looks for couples of elements that should be merged in order to maintain determinism, these new pairs are added to the *queue*. Merges continue until *queue* becomes empty, but if any merge fails, the complete chain of merges is undone. When all the pairs are considered algorithm finishes and the automaton associated with $M$ is the answer proposed.

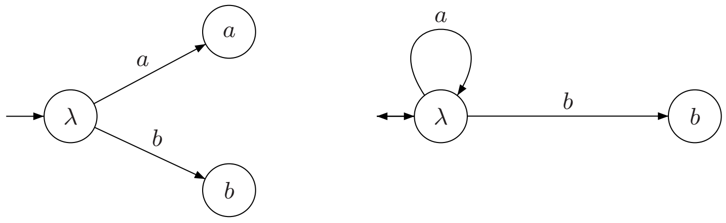


**Fig. 2.** $PTM(D_+ \cup D_-)$ with $\Phi(\lambda) = \uparrow$, $\Phi(a) = 1$ and $\Phi(b) = 0$ (on the left), and the resulting automaton from the RPNI algorithm for $D_+ = \{a\}$ and $D_- = \{b\}$ (on the right)

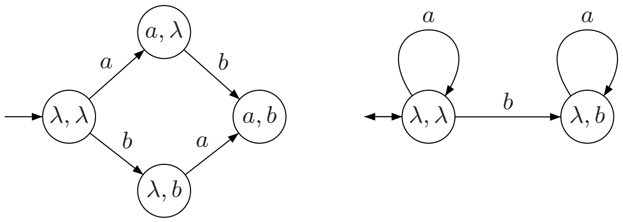


**Fig. 3.** $CPAM(D_+ \cup D_-)$ with $\Phi(\lambda, \lambda) = \uparrow$, $\Phi(a, \lambda) = 1$, $\Phi(\lambda, b) = 0$ and $\Phi(a, b) = \uparrow$, (on the left) and the resulting automaton from the CRPNI algorithm for $D_+ = \{a\}$ and $D_- = \{b\}$ (on the right)

Note that the order for merging states plays a crucial role in the algorithm. In our case, the order is always we compare any state with its predecessor by $<_{lex}$ and anyone of this elements one by one, in other words, if we have $\Sigma = \{a, b\}$, the merging order would be $aa$ with $a$, $bb$ with $b$, $aaa$ with $a$, ...

*Example 3.1.* We show in Figures 2 and 3 , $CPAM(D_+ \cup D_-)$, $PTM(D_+ \cup D_-)$ and the final result of the algorithms RPNI and CRPNI when we use them with samples $D_+ = a$ and $D_- = b$.

## 3.2 Convergence of *CRPNI*

Let $\Sigma = \{a_1, a_3, \ldots, a_n\}$ be an alphabet, let $L$ be a commutative language on $\Sigma$, and $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ the minimal commutative automaton of $L$.

Now, in order to show that the algorithm converges, it suffices to give a characteristic sample for the inference algorithm, and show than for it CRPNI algorithm infer the minimal deterministic automaton.

For the definition of the characteristic sample we define for $1 \leq i \leq n$ and $q^{a_i} \in Q_{a_i}$, the index $I(q^{a_i}) \geq 0$ such that $[a_i^{I(q^{a_i})}]_{\sim_L} = q^{a_i}$ and for any $j \leq 0$ with $[a^j]_{\sim_L} = q^{a_1}$, $I(q^{a_i}) \leq j$.

We define the characteristic sample $D = D_+ \cup D_-$ as a sample satisfying the following condition:

1. for $1 \leq i \leq n$, for any, $q_1^{a_i}, q_2^{a_i} \in Q_{a_1}$ with $q_1^{a_i} \neq q_2^{a_i}$, there exists $x \in D_+$ and $x \in D_-$ (or $x \in D_-$ and $x \in D_+$) such that $\pi_{a_i}(x) \geq I(q_1^{a_i})$, $\pi_{a_i}(y) \geq I(q_2^{a_i})$, $|\pi_{a_i}(x)| - I(q_1^{a_i}) = |\pi_{a_i}(y)| - I(q_2^{a_i})$ and for any $1 \leq j \leq n$ with $i \neq j$, $\pi_{a_j}(x) = \pi_{a_j}(y)$.
2. for $1 \leq i \leq n$, for any, $q_1^{a_i}, q_2^{a_i} \in Q_{a_1}$ such that $q_2^{a_i} \neq \delta(q_1^{a_i}, a_i)$, there exists $x \in D_+$ and $y \in D_-$ (or $x \in D_-$ and $y \in D_+$) such that $\pi_{a_i}(x) \geq I(q_1^{a_i})$, $\pi_{a_i}(y) \geq I(q_2^{a_i})$, $|\pi_{a_i}(x)| - (I(q_1^{a_i}) + 1) = |\pi_{a_i}(y)| - I(q_2^{a_i})$ and for any $1 \leq j \leq n$ with $i \neq j$, $\pi_{a_j}(x) = \pi_{a_j}(y)$.
3. for any $(q^{a_1}, \ldots, q^{a_n}) \in F$ there exists $x \in D_+$ such that $(q^{a_1}, \ldots, q^{a_n}) = ([\pi_{a_1}(x)]_{\sim_L}, \ldots, [\pi_{a_n}(x)]_{\sim_L})$.

**Proposition 4.** *CRPNI converges for the characteristic sample.*

*Proof.* Let again $\Sigma = \{a_1, a_3, \ldots, a_n\}$ be an alphabet, let $L$ be a commutative language on $\Sigma$, and $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ the minimal commutative automaton of $L$.

In order to show that the algorithm converges we show that the automaton we build at any moment is a subautomaton of the minimal commutative automaton. As the automaton that we are building is determined by $Q_{a_i}$ and $\delta_{a^i}$ for $1 \leq i \leq n$ and $F \subseteq Q_{a_1} \times \ldots \times Q_{a_n}$, it suffices to see that these elements are as the ones of the minimal commutative automaton in the part already learnt. We denote by $M = (Q', \Sigma, \Gamma, \delta', q_0', \Phi)$ the Moore machine that initially will be equal to $CPAM(D_+ \cup D_-) = (Q'', \Sigma, \Gamma, \delta'', q_0'', \Phi')$ and on which we will apply the algorithm. For any $l \geq 0$ and $1 \leq i \leq n$ such that $a^l \in Q''$ we will denote by $[a^l]_M$ the state of $Q'$ corresponding to all the states merged with $a^l$ in $M$.

For $1 \leq i \leq n$, we will denote by $K_{a_i} \subseteq Q_{a_1}$ (Kernel of $a_i$) to the set of states that have compared with all precedent states, and for $B_{a_i} \subseteq Q_{a_1}$ (Border of $a_i$) we will denote the set $\{q^{a_i} \in Q_{a_i} \mid$ there exists $q_2^{a_i}$ such that $\delta_{a_i}(q_2^{a_i}, a_i) = q^{a_i}$ and $q_2^{a_i} \in K\}$.

Now, it suffices to show that states and transition in $K_{a_i}$ belong to the minimal commutative automaton, and that the final states of the learnt automaton corresponds with the states of $\mathcal{A}$.

For the states, let $[a_i^j]_M, [a_i^k]_M \in K_{a_i}$ choosing $j$ and $k$ in such a way that $I([a_i^j]_{\sim_L}) = j$ and $I([a_i^k]_{\sim_L}) = k$ with $[a_i^j]_{\sim_L} \neq [a_i^k]_{\sim_L}$, then by condition 1 of the definition of the characteristic sample there exists $x \in D_+$ and $y \in D_-$ (or $x \in D_-$ and $y \in D_+$) such that $\delta(([\lambda]_M, \ldots, [\lambda]_M), x) = ([a_1^{m_1}]_M, \ldots, [a_i^j a_i^{m_i}]_M, \ldots, [a_n^{m_n}]_M)$, $\delta(([\lambda]_M, \ldots, [\lambda]_M), y) = ([a_1^{m_1}]_M, \ldots, [a_i^k a_i^{m_i}]_M, \ldots, [a_n^{m_n}]_M)$ and then necessarily $merge(M, [a_i^j]_M, [a_i^k]_K)$ fails. So, by the order of merging we have that all states in $K_{a_i}$ are equivalent to some state of $\mathcal{A}$.

For the transition, let $[a_i^j]_M, [a_i^k]_M \in K_{a_i}$ choosing $j$ and $k$ in such a way that $I([a_i^j]_{\sim_L}) = j$ and $I([a_i^k]_{\sim_L}) = k$ with $[a_i^k]_{\sim_L} \neq \delta([a_i^j]_{\sim_L}, a_i)$, then by condition 2 of the definition of the characteristic sample there exists $x \in D_+$ and $y \in D_-$ (or $x \in D_+$ and $y \in D_-$) such that $\delta(([\lambda]_M, \ldots, [\lambda]_M), x) = ([a_1^{m_1}]_M, \ldots, [a_i^j a_i^{m_i}]_M, \ldots, [a_n^{m_n}]_M)$, and $\delta(([\lambda]_M, \ldots, [\lambda]_M), y) = ([a_1^{m_1}]_M, \ldots, [a_i^k a_i^{m_i}]_M, \ldots, [a_n^{m_n}]_M)$ and then necessarily $merge(M, [a_i^j]_M, [a_i^k]_K)$ fails. Again by the order of merging we have that all transitions in $K_{a_i}$ are equivalent transition of $\mathcal{A}$.

Finally, since all states and transition in $K$ forms a subautomaton of $\mathcal{A}$, it suffices to see that at the end of the algorithm by condition 3 of the definition of the characteristic sample we have that for any $(q^{a_1}, \ldots, q^{a_n}) \in F$ there exists $x \in D_+$ such that for $(\pi_1(x), \ldots, \pi_n(x)) = ([q^{a_1}]_M, \ldots, [q^{a_n}]_M)$ which is equivalent to $([q^{a_1}]_{\sim_L}, \ldots, [q^{a_n}]]_{\sim_L})$ with $\Phi([q^{a_1}]_M, \ldots, [q^{a_n}]_M) = 1$ and then the algorithm identifies correctly $F$.

## 4 Experimental Results

The aim of this experimentation is to evaluate the performance of the *CRPNI* algorithm strategy with respect to classical inference methods such as redblue and RPNI when they are applied to the learning of regular commutative languages. We do not do an experiment of learning regular languages in general because CRPNI-algorithm is not able to learn general regular languages, and even the samples could be inconsistent, i.e. $D_+ = \{ab\}$ and $D_- = \{ba\}$ is an inconsistent sample, and in those cases the algorithms should not work. The criteria of comparison are the recognition rate on test samples and the average size of the hypothesis generated, the size of a model is its number of states.

The target languages are generated randomly, taking care of guarantee their complete commutativity between the symbols. The generation strategy consists on choose random transitions from each state previously generated and with each symbol. To control the difficulty degree of target languages is possible to change the number of different states available to build the target automaton. The experimentation consists of target languages on 2 and 3 symbols alphabet.

The corpus consist of several incremental sets of different samples, tagged for each target language. The size of the training set varies from 10 samples to 500 samples. The test set consists of 1000 samples different from training ones.

Table 1 shows the results of the first experiment. In this case, three algorithms are compared: RPNI, redblue and *CRPNI*. For each one were trained 200 regular commutative regular target languages which states number range between 4 and 30 states, their average size is 11.52 states. The corpus contains incremental training sets of 10, 20, 30, 40, 50, 100, 200, 300, 400 and 500 samples. The identification of each experiment shows the size of the training set, the percentage presented for each algorithm is the average of the recognition rate on the test set for the 200 target languages.

With respect to the size of the hypothesis proposed by the algorithms, Table 2 shows the behaviour of the three algorithms. The average of the states number for each of the 200 languages learned is presented.

The results on Table 1 and Table 2 shows a notorious improvement in recognition rate using the *CRPNI* with respect to the reference algorithms RPNI and redblue. With 100 training samples the new algorithm reaches more than 99% of accuracy and the size of its hypothesis is like the competitors or even smaller.

Table 3 shows the results of the second experiment, which trains bigger commutative regular target languages with two symbols alphabet. The states number of the target languages varies between 4 and 60 states, the average size is 57.46 states. Three algorithms are compared: RPNI, redblue and our proposal. The

**Table 1.** Recognition rates of *RPNI*, *redblue* and new algorithm in the first experiment

| id | RPNI | redblue | *CRPNI* |
|---|---|---|---|
| t10 | 69.74% | 67.77% | 83.74% |
| t20 | 75.00% | 70.22% | 90.43% |
| t30 | 78.18% | 75.36% | 93.30% |
| t40 | 79.62% | 77.60% | 95.75% |
| t50 | 82.20% | 80.30% | 98.12% |
| 100 | 87.71% | 86.46% | 99.69% |
| t200 | 91.49% | 91.25% | 99.89% |
| t300 | 94.38% | 93.66% | 99.95% |
| t400 | 95.23% | 94.74% | 99.96% |
| t500 | 95.70% | 96.07% | 99.97% |

**Table 2.** Average states number of *RPNI*, *redblue* and new algorithm in first experiment

| id | RPNI | redblue | *CRPNI* |
|---|---|---|---|
| t10 | 4.09 | 4.20 | 4.25 |
| t20 | 5.65 | 5.97 | 6.59 |
| t30 | 6.65 | 6.99 | 8.70 |
| t40 | 7.69 | 7.85 | 9.53 |
| t50 | 8.25 | 8.19 | 9.83 |
| t100 | 10.21 | 9.87 | 10.34 |
| t200 | 12.58 | 11.78 | 10.48 |
| t300 | 12.69 | 12.61 | 10.55 |
| t400 | 13.55 | 13.36 | 10.57 |
| t500 | 14.79 | 13.52 | 10.58 |

**Table 3.** Recognition rates of *RPNI*, *redblue* and new algorithm in second experiment

| id | RPNI | redblue | *CRPNI* |
|---|---|---|---|
| t10 | 70.88% | 70.92% | 78.55% |
| t50 | 71.73% | 71.12% | 90.73% |
| t100 | 73.09% | 72.99% | 96.72% |
| t200 | 74.19% | 74.64% | 99.01% |
| t300 | 75.94% | 76.48% | 99.52% |
| t400 | 78.30% | 78.96% | 99.72% |
| t500 | 80.23% | 81.05% | 99.78% |

corpus contains incremental training sets of 10, 50, 100, 200, 300, 400 and 500 samples. The identification of each experiment shows the size of the training set, the percentage presented for each algorithm is the average of the recognition rate on the test set for the 200 target languages.

With respect to the size of the hypothesis proposed by the algorithms, Table 4 shows the behaviour of the three algorithms. The average of the states number for each of the 200 languages learned is presented.

**Table 4.** Average states number of *RPNI*, *redblue* and new algorithm in third experiment

| id | RPNI | redblue | *CRPNI* |
|---|---|---|---|
| t10 | 4.12 | 4.16 | 5.59 |
| t50 | 11.16 | 11.15 | 29.51 |
| t100 | 17.35 | 16.56 | 40.7 |
| t200 | 28.10 | 26.51 | 47.05 |
| t300 | 37.26 | 34.21 | 49.15 |
| t400 | 43.90 | 40.08 | 50.03 |
| t500 | 50.14 | 45.11 | 50.45 |

**Table 5.** Recognition rates of *RPNI* and new algorithm in third experiment

| id | RPNI | *CRPNI* |
|---|---|---|
| t10 | 52.38% | 61.67% |
| t20 | 52.41% | 69.29% |
| t30 | 52.85% | 77.18% |
| t40 | 52.28% | 82.96% |
| t50 | 52.97% | 87.32% |
| t100 | 54.06% | 96.38% |
| t200 | 57.58% | 98.84% |
| t300 | 58.87% | 99.48% |
| t400 | 59.80% | 99.66% |
| t500 | 60.86% | 99.77% |

Second experiment shows, again, a clear superiority of *CRPNI* with respect to the reference ones, almost 20 points are the quantitative difference between them in recognition rate. Beside, the size of the hypothesis proposed becomes smaller than those of the reference algorithms as the size of the training set grows.

Table 5 shows the results of the third experiment. In this case, a three symbols alphabet is used. Two algorithms are compared: RPNI and our proposal; redBlue is not reported because previous experiments showed similar behaviour between RPNI and redblue. We trained 200 regular commutative regular target languages which states number range between 6 and 90 states, the average size is 35.23 states. The corpus contains incremental training sets of 10, 20, 30, 40, 50, 100, 200, 300, 400 and 500 samples. The identification of each experiment shows the size of the training set, the percentage presented for each algorithm is the average of the recognition rate on the test set for the 200 target languages.

With respect to the size of the hypothesis proposed by the algorithms, Table 6 shows the behaviour of the algorithms. The average of the states number for each of the 200 languages learned is presented.

In the third experiment the reference algorithm gets stick on a recognition rate of 60% while CRPNI reaches a recognition rate higher than 99% with 300 training samples. The size of the hypothesis of the new algorithm stabilises from 50 training samples on while RPNI ones grows until the last training set.

**Table 6.** Average states number of *RPNI* and new algorithm in third experiment

| id | RPNI | *CRPNI* |
|---|---|---|
| t10 | 4.60 | 8.09 |
| t20 | 6.56 | 17.47 |
| t30 | 8.31 | 26.24 |
| t40 | 9.98 | 33.01 |
| t50 | 11.41 | 36.14 |
| t100 | 18.24 | 35.37 |
| t200 | 28.57 | 35.69 |
| t300 | 37.99 | 34.51 |
| t400 | 46.11 | 34.58 |
| t500 | 54.29 | 34.62 |

Comparing results with $|\Sigma| = 2$ and $|\Sigma| = 3$ it is noticeable the increase in performance of CRPNI as soon as alphabet size grows. These results lead us to think that CRPNI could behave even better with bigger alphabets.

Although this experiments are suitable to be improved to make them even harder, this preliminary test is very promising about the utility of this new algorithm for the inference of commutative regular languages, not only because of the excellent recognition rates achieved but also because of the reasonable size of the hypothesis obtained.

## 5 Conclusions

In this work we study the problem of inferring the class of regular commutative languages. After showing that they are not inferable from positive data we show that some improvement in its inferring from positive and negative samples can be done. For this purpose we give the CRPNI (commutative regular positive negative inference) algorithm. These properties lead us to define commutative deterministic automata and commutative Moore machines for the algorithm. Finally, by an experimentation we also show that CRPNI algorithm has an excellent behaviour in practice. This shows that this kind of works can be useful for real problems.

## References

1. Ruiz, J., Cano, A., García, P.: Inferring subclasses of regular languages faster using RPNI and forbidden configurations. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 28–36. Springer, Heidelberg (2002)
2. Agluin, D.: Inductive inference of formal languages from positive data. Information and Control 45(2), 117–135 (1980)

3. Alquézar, R., Sanfeliu, A.: Incremental grammatical inference from positive and negative data using unbiased finite state automata. In: Shape, Structure and Pattern Recogniton, Proc. Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR 1994, pp. 291–300. World Scientific, Singopore (1995)
4. Cichelo, O., Kremer, S.C.: Inducing grammars from sparce data sets: A survey of algorithms and results. Journal of Machine Learning Research 4, 603–632 (2003)
5. Clark, A., Florêncio, C.C., Watkins, C., Serayet, M.: Planar languages and learnability. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 148–160. Springer, Heidelberg (2006)
6. Coste, F., Fredouille, D.: Efficient ambiguity detection in C-NFA. In: Oliveira, A.L. (ed.) ICGI 2000. LNCS (LNAI), vol. 1891, pp. 25–38. Springer, Heidelberg (2000)
7. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
8. Hopcroft, J., Ullman, J.: Introduction to automata theory, languages and computation. Addison-Wesley, Reading (1980)
9. Pearlmutter, B.A., Lang, K.J., Price, R.A.: Results of the abbadingo one DFA learning competition and a new evidence state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 1–12. Springer, Heidelberg (1998)
10. Lang, K.J.: Random dfa's can be approximately learned from sparse uniform. In: Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, pp. 45–52 (1992)
11. Oncina, J., Garcia, P.: Inferring regular languages in polynomial updated time. In: Pattern Recognition and Image Analysis (1992)
12. Pin, J.-E.: Varieties of formal languages. North Oxford, London (1986) (Traduction of Variétés de langages formels)
13. Pin, J.-É.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of formal languages, ch. 10, vol. 1, pp. 679–746. Springer, Heidelberg (1997)

# Learning Left-to-Right and Right-to-Left Iterative Languages

Jeffrey Heinz

University of Delaware, Newark DE 19716, USA
heinz@udel.edu

**Abstract.** The left-to-right and right-to-left iterative languages are previously unnoticed subclasses of the regular languages of infinite size that are identifiable in the limit from positive data. Essentially, these language classes are the ones obtained by merging final states in a prefix tree and initial states in a suffix tree of the observed sample, respectively. Strikingly, these classes are also transparently related to the zero-reversible languages because some algorithms that learn them differ minimally from the ZR algorithm given in Angluin (1982). Second, they are part of the answer to the challenge provided by Muggleton (1990), who proposed mapping the space of language classes obtainable by a general state-merging algorithm IM1. Third, these classes are relevant to a hypothesis of how children can acquire sound patterns of their language—in particular, the hypothesis that all phonotactic patterns found in natural language are neighborhood-distinct (Heinz 2007).

## 1 Introduction

One motivation behind the learning paradigm known as *identification in the limit from positive data* [1] is the observation that children encounter only positive examples of natural language [2]. Gold's (1967) result that no class of languages including all finite languages plus one infinite language is learnable this way—and hence the regular, context-free, and context-sensitive languages are not either—launched research to find learnable subclasses which crosscut the Chomsky Hierarchy [3,4,5,6,7,8,9,10,11,12,13]. This approach is also justifiable from a linguistic perspective because language typologists repeatedly observe that the extensive variation that exists in natural languages appears to be limited, though stating exact universals is difficult [14,15,16].

One focus of the grammatical inference community is finding learnable classes of languages which reach higher regions of the Chomsky Hierarchy [11,12,13]. This is partly because the most complex known natural language patterns are at least mildly context-sensitive [17,18], and partly because of the technical challenge. However, the hypothesis that all phonological patterns—i.e. sound patterns—are regular is well-supported [19,20,21]. In other words, although sentence well-formedness seems to necessitate mildly-context sensitive computations over words, word well-formedness seems only to require regular computations over individual sounds. Thus it remains an important open question what subclasses of the regular languages (if any) are identifiable in the limit from positive data and which properly include word well-formedness patterns observed cross-linguistically (see also [22]). This question is especially interesting in light of acquisition research which indicates infants acquire many word well-formedness patterns within one year of birth [23,24,25]; i.e. before they can talk.

This paper addresses this question by introducing and characterizing the left-to-right iterative (LRI) and right-to-left iterative (RLI) languages, which are relevant to natural language word well-formedness patterns (henceforth *phonotactic patterns*). Essentially, these language classes are the ones obtained by merging final states in a prefix tree and initial states in a suffix tree of the observed sample, respectively. They are interesting for at least four reasons.

First, [7] proposes a general state-merging algorithm IM1 for learning subclasses of the regular languages. Essentially, IM1 returns a nondeterministic finite-state acceptor by merging states according to some state equivalence criteria. To my knowledge, no systematic study of the language classes obtained in this way has been undertaken. This paper contributes to this research program, whose known results and open questions are summarized in §3. Second, this work introduces the concept of head-canonical acceptor which is the smallest reverse deterministic acceptor for a regular language (useful in understanding RLI languages). Third, these two language classes are closely related to—but incomparable with—the zero-reversible languages since algorithms that identify them are essentially the same as algorithm ZR in [5] with one line removed.

Finally, LRI and RLI languages are relevant to phonotactic patterns. When hundreds of phonotactic patterns are collected and placed in the Chomsky Hierarchy, they all fall into the class of the regular languages [21]. More specifically, they are Noncounting, a class known not to be identifiable in limit from positive data [26,22]. [21] hypothesizes all phonotactic patterns belong to a smaller class called neighborhood-distinct (defined in §3), which crosscuts the Subregular Hierarchy. Neighborhood-distinctness is a complex property, making it difficult to analyze. LRI and RLI languages are properly thought of as basic components of this more complex class, and therefore this paper also makes a step towards understanding neighborhood-distinctness.

This paper is organized as follows. §2 introduces notation and definitions. §3 discusses a general state-merging algorithm due to [7]. §4 presents the neighborhood-distinct hypothesis. §5 characterizes the LRI languages in automata- and language-theoretic terms and gives a procedure which identifies this class in the limit from positive data, and §6 does the same for RLI languages. §7 summarizes the contributions and open questions.

## 2 Preliminaries

This section establishes notation and basic definitions. A set $\pi$ of nonempty subsets of $S$ is a *partition* of $S$ iff the elements of $\pi$ are pairwise disjoint and their union equals $S$. Each *block* in $\pi$ containing $x \in S$ is denoted $[x]_\pi$. A partition

$\pi$ *refines* another partition $\pi'$ iff every block of $\pi$' is a union of blocks of $\pi$. If $\pi$ is a partition of a set $S$ and $S' \subseteq S$, then the *restriction* of $\pi$ to $S'$ is the partition $\pi'$ consisting of all sets $B'$ that are nonempty and are the intersection of $S'$ and some block of $\pi$. The *trivial partition* is the unique partition where $|\pi| = |S|$.

### 2.1 Strings and Languages

$\Sigma$ denotes a fixed finite set of symbols, the *alphabet*. Let $\Sigma^n$, $\Sigma^{\leq n}$, $\Sigma^*$ denote all sequences over this alphabet of length $n$, of length less than or equal to $n$, and of any finite length, respectively. For sequence $s$, $range(s)$ is the set of elements in $s$. $\lambda$ denotes the empty string and $|w|$ denotes the length of string $w$. The reverse of string $u$ is denoted $u^r$. A string $u$ is a *prefix* (*suffix*) of $w$ iff there exists $v$ in $\Sigma^*$ such that $w = uv$ ($w = vu$).

A language $L$ is a subset of $\Sigma^*$. $L^r = \{u^r : u \in L\}$. $L_1 \cdot L_2 = \{uv : u \in L_1 \text{ and } v \in L_2\}$. Like strings above, let $L^0 = \{\lambda\}$, $L^{n+1} = L^n \cdot L$, and $L^* = \bigcup_{n \in \mathbb{N}} L^n$. Let $L^{|k|}, L^{\leq |k|}$ denote all strings in $L$ with length exactly $k$ and length less than or equal to $k$, respectively. The *length* of a finite language $L$ is $length(L) = \Sigma_{w \in L} |w|$. The *prefixes* of language $L$ are given by $Pref(L) = \{u : \exists v \text{ so that } uv \in L\}$. The *suffixes* of a language are defined as $Suff(L) = \{u : \exists v \text{ so that } vu \in L\}$. The *tails* of $w$ given $L$, is denoted by $T_L(w) = \{u : wu \in L\}$. Also, the *heads*, of $w$ given $L$, is denoted by $H_L(w) = \{u : uw \in L\}$. A language $L$ naturally induces partitions $\pi_{TL}$: $[u]_{\pi_{TL}} = [v]_{\pi_{TL}}$ iff $T_L(u) = T_L(v)$ and $\pi_{HL}$: $[u]_{\pi_{HL}} = [v]_{\pi_{HL}}$ iff $H_L(u) = H_L(v)$. The Myhill-Nerode theorem states that a language is *regular* iff $\pi_{TL}$ is finite.

### 2.2 Identification in the Limit

A positive text $S$ of a language $L$ is an infinite sequence such that $range(S) = L$. $S_t$ denotes the first $t$ elements of $S$. A learner $\phi$ is an algorithm which maps finite sequences of words to grammars. The learner $\phi$ *identifies a class of languages* $\mathcal{L}$ *in the limit from positive data* iff for all $L \in \mathcal{L}$, for all positive texts $S$ for $L$, there is some $i \in \mathbb{N}$ such that for all $j > i$, $\phi(S_j)$ is a grammar recognizing $L$. A language class with such a $\phi$ is *identifiable in the limit from positive data*. If for each $L \in \mathcal{L}$, there is a finite language $S_L \subseteq L$ such that for all other $L' \in \mathcal{L}$ which contain $S_L$, $L \subseteq L'$, then $\mathcal{L}$ is identifiable in the limit from positive data and $S_L$ is called a *characteristic sample* for $L$ in $\mathcal{L}$ [4,5].

### 2.3 Finite State Acceptors

For a given $\Sigma$, a finite-state acceptor (FSA) is a quadruple $A = (Q, I, F, \delta)$ such that $Q$ is finite, $I$ and $F$ are subsets of $Q$ and $\delta : Q \times \Sigma \rightarrow 2^Q$. $\delta$ is extended recursively so that $\delta : 2^Q \times \Sigma^* \rightarrow 2^Q$. The language of an acceptor $A$ is $L(A) = \{w \in \Sigma^* : \delta(I, w) \cap F \neq \emptyset\}$. Languages recognizable by FSAs are regular. We assume familiarity with regular expressions, which also define regular languages.[1] [27] provides other well-known characterizations of this class.

[1] See [34] for the regular expression notation used in this paper.

Consider two acceptors $A = (Q, I, F, \delta)$ and $A' = (Q', I', F', \delta')$. Acceptors $A$ and $A'$ are *equivalent* iff $L(A) = L(A')$. $A$ and $A'$ are *isomorphic* iff there is a bijection $h$ from $Q$ to $Q'$ such that $h(I) = I'$, $h(F) = F'$, and for all $q \in Q$ and $a \in \Sigma$ it is the case that $h(\delta(q, a)) = \delta'(h(q), a)$. $A'$ is a *subacceptor* of $A$ iff $Q' \subseteq Q$, $I' \subseteq I$, $F' \subseteq F$, and for every $q' \in Q'$ and $a \in \Sigma$, $\delta'(q', a) \subseteq \delta(q', a)$. It follows that if $A'$ is a subacceptor of $A$ then $L(A') \subseteq L(A)$. The reverse of $A$ is $A^r = (Q, F, I, \delta^r)$, where $\delta^r(q, a) = \{q' : q \in \delta(q', a)\}$ for all $a \in \Sigma, q \in Q$.

An acceptor $A = (Q, I, F, \delta)$ is *forward deterministic* iff $|I| \leq 1$ and each $q \in Q$ has at most one $b$-successor for all $b \in \Sigma$. $A$ is *reverse deterministic* iff $|F| \leq 1$ and each $q \in Q$ has at most one $b$-predecessor for all $b \in \Sigma$. An acceptor which is both forward and reverse deterministic is called *zero-reversible* [5].

An acceptor is *trimmed* iff for all $q \in Q$, there are $u, v \in \Sigma^*$ such that $q \in \delta(I, u)$ and $\delta(q, v) \cap F \neq \emptyset$. An acceptor is *cyclic* iff it is trimmed and has at least one loop. An acceptor which is not cyclic is *acyclic*. Two important acyclic acceptors are *prefix* and *suffix trees*. The prefix (suffix) tree of a finite language $S$ is forward (backward) deterministic, and is defined below.

| PT(S) | ST(S) |
|---|---|
| $Q = Pref(S)$ | $Q = Suff(S)$ |
| $I = \{\lambda\}$ | $I = S$ |
| $F = S$ | $F = \{\lambda\}$ |
| $\delta(u, a) = ua$ iff $u, ua \in Q$ | $\delta(au, a) = u$ iff $u, ua \in Q$ |

These acceptors are not mirror images of each other, though they are equivalent. It is easy to show for any finite language $S$, $PT^r(S)$ is isomorphic to $ST(S^r)$.

The *tail-canonical acceptor* for a regular language $L$ is denoted $A_T(L)$ and is defined below. Acceptors isomorphic to the tail-canonical acceptor are *tail-canonical*. For a regular language $L$, a tail-canonical acceptor is the forward deterministic acceptor with the fewest states.The *head-canonical acceptor* for $L$ is $A_H(L)$ is also defined below. Acceptors isomorphic to the head-canonical acceptor are called *head-canonical*. A head-canonical acceptor is the acceptor with the fewest states for a regular language $L$ that is backward deterministic.

| $A_T(L)$ | $A_H(L)$ |
|---|---|
| $Q = \{T_L(u) : u \in Pref(L)\}$ | $Q = \{H_L(u) : u \in Suff(L)\}$ |
| $I = \{T_L(\lambda)\}$ | $I = \{H_L(w) : w \in L\}$ |
| $F = \{T_L(w) : w \in L\}$ | $F = \{H_L(\lambda)\}$ |
| $\delta(T_L(u), a) = T_L(ua)$ iff $u, ua \in Pref(L)$ | $\delta(H_L(au), a) = H_L(u)$ iff $u, au \in Suff(L)$ |

Theorem 1 shows how one can obtain an equivalent head canonical acceptor from a tail canonical one: reverse, determinize, minimize, and reverse again.

**Theorem 1.** *Let $L$ be a regular language. Then $A^r_H(L)$ is isomorphic to $A_T(L^r)$.*

The proof is omitted (see [21]), but the needed bijection is $h(L) = L^r$. Head canonical acceptors allow another definition of zero-reversible languages: they are those languages whose tail and head canonical acceptors are isomorphic.

The following notions are used in §§3-6. The *k-leaders* of a state $q$ are denoted $I_k(q) = \{u \in \Sigma^{\leq k} : \exists q' \in Q$ such that $q \in \delta(q', u)\}$. The *k-followers* of a state are denoted $O_k(q) = \{u \in \Sigma^{\leq k} : \exists q' \in Q$ such that $q \in \delta(q', u)\}$. ($I$ and $O$ invoke *incoming* and *outgoing*, respectively.) The function $final(q) = \mathbb{f}$ if $q \in F$, else $q$. The function $nonfinal(q) = \mathbb{nf}$ if $q \notin F$, else $q$. The function $start(q) = \mathbb{s}$ if $q \in I$, else $q$. The function $nonstart(q) = \mathbb{ns}$ if $q \notin I$, else $q$. For any $q \in Q$, $b \in \Sigma$, the *b-successors* of $q$ are $\delta(q, b)$ and the *b-predecessors* of $q$ are $\delta^r(q, b)$.

For some acceptor $A = (Q, I, F, \delta)$, a sequence $q_0 q_1 \dots q_k$ is a *path* in $A$ iff for all $0 \leq i \leq k-1$, there is $a \in \Sigma$ such that $q_{i+1} \in \delta(q_i, a)$. Paths $q_0 q_1 \dots q_k$ such that $q_0 = q_k$ and for $1 \leq i \leq k-1$, $q_i \neq q_0$ are called *loops*. Let $loops(A)$ denote the set of loops in $A$. Also, if $p = q_0 q_1 \dots q_k$ is a path then $strings(p) = \{u : u = a_0 a_1 \dots a_{k-1}$ where for all $0 \leq i \leq k-1$, $q_{i+1} \in \delta(q_i, a_i)\}$.

## 3 Partitioning Acceptors

Let $A = (Q, I, F, \delta)$ be any acceptor. Any partition $\pi$ of $Q$, defines another acceptor $A/\pi = (Q', I', F', \delta')$ defined as follows:

$$\begin{aligned} Q' &= \{B : [q]_\pi \text{ such that } q \in Q\} \\ I' &= \{B : [q]_\pi \text{ such that } q \in I\} \\ F' &= \{B : [q]_\pi \text{ such that } q \in F\} \\ \delta'([q]_\pi, a) &= \{[q']_\pi : q' \in \delta(q, a)\} \end{aligned}$$

For any acceptor $A$ and $\pi$ over the states of $A$ it follows that if $p \in \delta(q, u)$ then $[p]_\pi \in \delta'([q]_\pi, u)$. Hence $L(A/\pi)$ includes all strings in $L(A)$, possibly more.

*Remark 1.* For any $A$ and $\pi$ over $Q$, if $p = q_0 \dots q_k$ is a path in $A$ which is not a loop and $[q_0]_\pi = [q_k]_\pi$ then $p' = [q_0]_\pi \dots [q_k]_\pi$ is a *new loop* in $A/\pi$.

*Remark 2.* Consider $PT(S)$ and any $\pi$ over the states of $PT$, and consider any state $B$ in $PT(S)/\pi$. If $|B| = 1$ then $I_1(B) \leq 1$. In other words, states which are not merged with others have at most one 1-leader. Similarly for any state $B$ in $ST(S)/\pi$, if $|B| \leq 1$ then $O_1(B) \leq 1$.

It is also known that if a sample of words of some regular language $L$ is sufficient—that is if generated by $A_T(L)$ then every transition in $A_T(L)$ would be exercised—then there exists some partition $\pi_T$ of $PT(S)$ such that $PT(S)/\pi_T$ is isomorphic to $A_T(L)$ [5]. Similarly, it follows that if generating $S$ exercises every transition in the head canonical acceptor, that there is some some partition $\pi_H$ of $ST(S)$ such that $ST(S)/\pi_H$ is isomorphic to $A_H(L)$.

The merging procedure above is independent of the decision of state-equivalence. The latter aspect determines the generalization strategy of the learner, and ultimately the class of languages that can be learned (if any). [7] proposes a general state-merging algorithm IM1 based partly on this observation, and functions $f : Q \to A$, which naturally induce a partition $\pi_f$ over $Q$: for all $q_1, q_2 \in Q$, $[q_1]_{\pi_f} = [q_2]_{\pi_f}$ iff $f(q_1) = f(q_2)$. IM1 computes $PT(S)/\pi_f$.

Here IM1 is generalized to compute $M(S)/\pi_f$, where $M(S)$ is any well-defined FSA recognizing the sample. This study limits $M$ to prefix and suffix trees. It follows that the class of languages obtained by an algorithm which computes $M(S)/\pi_f$ depends not only on $f$ but on $M$ as well.

For example, if $M = PT$ and $f = I_k$ then the language class obtained is the Locally $(k + 1)$ Testable in the Strict Sense (LTSS) [6]. If only start states are merged in the prefix tree (i.e $M = PT$ and $f = start$), or only final states in the suffix tree ($M = ST$ and $f = final$), it is easy to see that $\mathcal{L}_{fin}$ is the class of languages obtained (since no states are actually merged). In §§5-6 it is shown that the left-to-right iterative languages ($LRI$) and right-to-left iterative languages ($RLI$) are the language classes obtained by merging final states in the prefix tree and start states in the suffix tree, respectively. Table 1 summarizes the known language classes identifiable in the limit from positive data by varying parameters $M$ and $f$. The fact that language classes of some cells are the reverse of language classes of other cells reveals an underlying algebra, whose exact properties await future discovery.

## 4 The Neighborhood-Distinct Hypothesis

Part of the motivation for filling in Table 1 comes from the hypothesis that all natural language phonotactic patterns are *1-1 neighborhood-distinct* [21]. The *j-k neighborhood* of a state is the tuple

$$nd_k^j(q) = (I_j(q), O_k(q), [\mathbf{q} \in \mathbf{F}], [\mathbf{q} \in \mathbf{I}])$$

For example, the 1-1 neighborhood of state 5 in Fig. 1 is $(\{g\}, \{h, i\}, 0, 0)$. An acceptor is *j-k neighborhood-distinct* iff every state has a unique j-k neighborhood. The languages of such acceptors are called *j-k neighborhood-distinct* (j-k ND).

[21] shows three main classes of phonotactic patterns—patterns over adjacent segments, long distance patterns such as vowel and consonantal harmony, and rhythmic patterns—are 1-1 ND. These patterns crosscut the Subregular Hierarchy. For example, Navajo and Sarcee have sibilant harmony patterns [28]. In Navajo, this pattern is symmetric: the sibilant sound [s] may not precede [ʃ] (sounds *s* and *sh*, respectively) in a word, even if they are separated by arbitrarily many other sounds, and vice versa. In Sarcee, the pattern is asymmetric: [ʃ] may precede [s] in a word, but [s] cannot precede [ʃ]. The Navajo pattern is Locally 1-Testable since one can decide if a string obeys the sibilant harmony pattern by

**Table 1.** Language Classes Obtained by Merging States in Prefix and Suffix Trees

| $f$ | $PT(S)/\pi_f$ | $ST(S)/\pi_f$ |
|---|---|---|
| $I_k$ | $(k+1)$ LTSS | ? |
| $O_k$ | ? | $(k+1)$ LTSS |
| $final$ | $LRI$ | $\mathcal{L}_{fin}$ |
| $start$ | $\mathcal{L}_{fin}$ | $RLI$ |
| $nonfinal$ | ? | $\{L_1^* \cdot L_2 : L_1, L_2 \subseteq \Sigma^1\}$ |
| $nonstart$ | $\{L_1 \cdot L_2^* : L_1, L_2 \subseteq \Sigma^1\}$ | ? |

checking whether [s] and [ʃ] are both present in the string. This procedure does not work for Sarcee since the order matters; hence it is Noncounting (also called Locally Testable with Order [26]).[2] These patterns are shown in Table 2 where C represents any consonant except sibilants, *s* sibilants like [s], *V* any vowel, and ʃ sibilants like [ʃ].

More examples comes from the different kinds of ways languages stress syllables in words [29]. In Sierra Miwok, main stress falls on the initial syllable if it is 'heavy', else on the peninitial syllable.[3] Secondary stress falls on all other heavy syllables. On the other hand, in Kwakwala, main stress falls on the leftmost 'heavy' syllable, but if there are none, on the final syllable. Patterns like Sierra Miwok are called *bounded* and are mostly 3-LTSS. Patterns like Kwakwala are called *unbounded* and Noncounting. Table 2 shows regular expressions for these patterns. The symbols *L* and *H* indicate light and heavy syllables, and acute and grave accents main and secondary stress, respectively. It is easy to verify that the examples in Table 2 are all 1-1 ND by drawing acceptors for them.

**Table 2.** Phonotactic Patterns

| Pattern | Example Language | Regular Expression |
|---|---|---|
| Symmetric Harmony | Navajo | (C+V+ʃ)*+(C+V+s)* |
| Asymmetric Harmony | Sarcee | (C+V+ʃ)*(C+V+s)$^*$ |
| Bounded Stress | Sierra Miwok | (LH́ + LĹ + H́)(H̀+L)* |
| Unbounded Stress | Kwakwala | (L*H́(H+L)*) + (L*L) |

There is currently no language-theoretic characterization of the j-k ND class due partly to its complexity. Also, [30,21,31] present an algorithm like IM1 which returns the intersection of the acceptors obtained by merging same-1-1-neighborhood states in prefix and suffix trees of the observed sample. While this algorithm appears to identify many 1-1 ND patterns (including almost all attested phonotactic patterns), it does not identify the class.[4] It remains an open question exactly what class is identified by this procedure. The line of research here aims to understand the primitive components that make up the neighborhood (see Table 1). The idea is that this language class is some composition of language classes obtained by simpler learners like the ones in Table 1 acting in concert (cf. [33]). Classes LRI and RLI are a small step towards this goal since

[2] [21] defines the precedence languages which contain long-distance harmony patterns. This class is identifiable in the limit from positive data with a string extension learner.

[3] A heavy syllable is typically more sonorous than a light syllable. Languages may make a heavy/light syllable distinction in different ways. See [29].

[4] For fixed j-k, the j-k ND class is finite, and so there are many learners which can learn this class [32]. However it is interesting to consider learners which generalize on the basis of hypothesized universal properties of natural language patterns, see [16] for discussion.

they relate to the indicator functions $[\mathbf{q} \in \mathbf{F}]$ and $[\mathbf{q} \in \mathbf{I}]$, which are boolean compositions of the functions $final$, $nonfinal$ and $start$, $nonstart$, respectively.

## 5 Left-to-Right Iterative Languages

LRI languages are defined as the intersection of the following two classes:

1. $\mathcal{L}_{1fd} = \{L : \text{whenever } u, v \in L, T_L(u) = T_L(v)\}$.
2. $\mathcal{L}_{LL^*fin} = \{L_1 \cdot L_2^* : L_1, L_2 \in \mathcal{L}_{fin}\}$

*LRI* languages are so named because words fix some strings to the left edge and then may iterate other strings rightward. As shown below, *LRI* languages are exactly the ones recognizable by acceptors which are forward deterministic, have at most one final state, and whose loops, if there are any, pass through the final state, if there is one. A schematic is given in Fig. 1.

Additionally, *LRI* is identifiable in the limit by a learner which computes $PT(S)/\pi_{final}$. Although this acceptor is not necessarily deterministic, it has one final state, all of its loops pass through this final state, and it can be made deterministic without altering those properties or the language recognized.

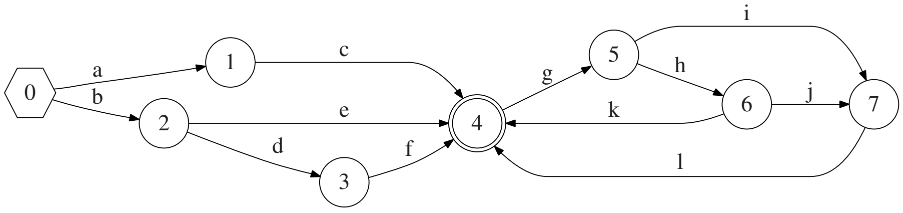


**Fig. 1.** Schematic of acceptors recognizing *LRI* languages

Call the class of languages recognized by acceptors which are forward deterministic with at most one final state *1-final-deterministic* and denote this class with $\mathcal{L}_{1fd}$. The proof of Theorem 2 is straightforward and thus omitted.

**Theorem 2.** *L is 1-final-deterministic iff whenever* $u, v \in L$, $T_L(u) = T_L(v)$.

Note that neither $\mathcal{L}_{LL^*fin}$ nor $\mathcal{L}_{1fd}$ are identifiable in the limit from positive data. $\mathcal{L}_{LL^*fin}$ contains all finite languages and at least one infinite language. As for $\mathcal{L}_{1fd}$, a limit point proof [32] establishes this claim, which is sketched here. Since $\{abc\}$, $\{abc, abbc\}$, $\{abc, abbc, abbbc\}$, $\ldots ab^*c$ all belong to $\mathcal{L}_{1fd}$, no learner can identify this subset in the limit from positive data—and hence not $\mathcal{L}_{1fd}$. Although neither $\mathcal{L}_{LL^*fin}$ nor $\mathcal{L}_{1fd}$ is identifiable in the limit, their intersection (*LRI*) is. Below is an automata-theoretic characterization of *LRI*.

**Theorem 3.** $L \in LRI$ *iff (1)* $A_T(L) = (Q, I, F, \delta)$ *is 1-final-deterministic and (2) if L is infinite, then every loop in* $A_T(L)$ *passes through the final state, i.e.* $F \subseteq \bigcap_{p \in loops(A)} range(p)$.

*Proof.* Consider any $L \in LRI$. Both directions of (1) follow from Theorem 2. Now assume $L$ is infinite and let $q_f$ denote the unique final state. Note $L = L_1 \cdot L_2^*$ where $L_1, L_2 \in \mathcal{L}_{fin}$. Suppose there is a loop $p = q_0 q_1 \dots\ q_k q_0$ in $A$ such that no $q_i = q_f$. Let $v \in strings(p)$. Since $A_T(L)$ is trimmed, there are $u, w$ such that $\delta(I, u) = q_0$ and $\delta(q_0, w) = q_f$. It follows that $uv^*w \subseteq\ L$. Since $L = L_1 \cdot L_2^*$ and since $p$ does not contain $q_f$, either $uv^* \subseteq L_1$ or $v^*w \subseteq L_2$. But this is a contradiction, since $L_1, L_2 \in \mathcal{L}_{fin}$. Thus every loop must pass through $q_f$. It remains to be shown that if all loops pass through the unique final state of $A_T(L)$, that $L = L_1 \cdot L_2^*$, $L_1, L_2 \in \mathcal{L}_{fin}$. It can be shown that $L_1$ is the language recognized by the largest trimmed acyclic subacceptor of $A_T(L)$ and $L_2$ is the union of $strings(p)$ for all loops $p = q_f q_1 \dots q_k q_f$. □

*Remark 3.* It follows if $L \in LRI$ then the language of any subacceptor of $A_T(L)$ is also left-to-right iterative.

*Remark 4.* In terms of regular expressions, it follows that a language $L$ belongs to LRI if $L = (a_0 + a_1 + \dots a_n)(b_0 + b_1 + \dots b_m)^*$ for $n, m \in \mathbb{N}$, $a_i, b_i \in \Sigma^*$ and for all $i, j$, no $a_i$ ($b_i$) is a proper prefix of $a_j$ ($b_j$).

With Theorem 3 and Remark 4 it is easy to see that the stress pattern of Miwok is in LRI, but not the other patterns in Table 2. LRI languages do not appear to characterize phonotactic patterns in general, even though they are related to them via the composition which constructs the neighborhood.

Also, it can be shown that *LRI* languages are incomparable with the ZR languages. In Fig. 1 if $j = i$ then the acceptor still accepts a language in LRI, but it is no longer backward deterministic and therefore its language is not in ZR. Similarly if instead we add a transition from state 1 to itself labeled $g$, then the language of the acceptor belongs to ZR, but not LRI.

The next lemma illustrates how generalization takes place, and is used to establish the fact that every language in *LRI* has a characteristic sample.

**Lemma 1.** *Let $L \in \mathcal{L}_{1fd}$ and let $x, y_i \in \Sigma^*$ for $0 \le i \le k$ such that $x, xy_i \in L$. Then $x(y_1 + y_2 + \dots + y_k)^* \subseteq\ L$.*

*Proof.* For some $k \in \mathbb{N}$, let $x, xy_1, xy_2, \dots xy_k \in L$. By induction on $n$, it is shown $x(y_1 + y_2 + \dots + y_k)^n \subseteq L$. Clearly when $n = 0$, $x \in L$. Now assume for some $n \in \mathbb{N}$, if $x, xy_1, xy_2, \dots xy_k \in L$ then $x(y_1 + y_2 + \dots + y_k)^n \subseteq L$. It remains to be shown that for all $1 \le i \le k$, $x(y_1 + y_2 + \dots + y_k)^n y_i \subseteq L$. For any $w \in L$, $y_i \in T_L(w)$ because $x, xy_i \in L$ and $L \in \mathcal{L}_{1fd}$ so by Theorem 2, $T_L(w) = T_L(x)$. By the inductive hypothesis, for all $w \in x(y_1 + y_2 + \dots + y_k)^n$, $w \in L$ and so therefore $wy_i \in L$. It follows that $x(y_1 + y_2 + \dots + y_n)^{n+1} \subseteq L$. □

**Theorem 4.** *For $L \in LRI$, there exists a characteristic sample $S_L$.*

*Proof.* For any $L \in LRI$, let $L_1, L_2 \in \mathcal{L}_{fin}$ such that $L = L_1 \cdot L_2^*$. Then $S = L_1 \cup L_1 \cdot L_2$ is characteristic. Note $S$ is finite. Let $L' \in LRI$ containing $S$. Consider any $w \in L$. It is sufficient to show that $w \in L'$. Since $w \in L$ and $L \in LRI$, there is some $k \in \mathbb{N}$ such that $w = xy_1y_2 \dots y_k$ where $x \in L_1$ and for any $1 \le i \le k$, $y_i \in L_2$. It is also the case that $x, xy_i \in S$. Since $S \subseteq L'$, $x(y_1 + y_2 + \dots + y_k)^* \subseteq\ L'$ by Lemma 1. Clearly $w \in x(y_1 + y_2 + \dots + y_k)^*$ and thus is in $L'$. □

As an example, if $L = L_1 \cdot L_2^*$ is in LRI and $L_1 = \{u, v\}$ and $L_2 = \{x, y, z\}$ then $S_L = \{u, v, ux, uy, uz, vx, vy, vz\}$. Since a characteristic sample $S_L$ for any $L \in LRI$ exists, a learner guessing $L$ after exposure to $S_L$ is picking the smallest $LRI$ language consistent with $S$.

For all $L$ in LRI, the size of $S_L$ grows polynomially with respect to the size of $A_T(L)$ (usually measured in states, see [35]). This is primarily because $L = L_1 \cdot L_2$ where $L_1$ and $L_2$ are finite languages, and thus the size of $A_T(L)$ is in the worst case approximates $length(L_1) + length(L_2)$. Now the length of $S_L$ equals $(|L_2|+1)length(L_1)+|L_1|length(L_2)$. Since for finite $L$, $|L| \leq length(L)+1$, the size of $S_L$ is bounded by a quadratic function over $length(L_1)$ and $length(L_2)$.

$PT(S)/\pi_{final}$ is not necessarily deterministic. We show that application of S-UPDATE, an algorithm which merges states that are b-successors of some state, returns an equivalent acceptor which is deterministic.

**Algorithm 1.** Pseudo-code for S-UPDATE (forward determinize)

```
Input: an acceptor $A$.
Output: a forward deterministic acceptor $A'$.
Initialization
Let $A_0 = (Q_0, I_0, F_0, \delta_0)$, $\pi_0$ the trivial partition of $Q_0$, and $i = 0$.
Let LIST contain all pairs $(q_1, q_2)$ such that $q_1$ and $q_2$ are distinct $b$-successors of
some $q_0 \in Q$ for all $b \in \Sigma$.
Merging
while LIST $\neq \emptyset$ do
    Remove some element $(q_1, q_2)$ from LIST.
    Let $\pi_{i+1}$ be the one obtained by merging blocks $[q_1]_{\pi_i}$ and $[q_2]_{\pi_i}$
    For all $[q], [r], [s]$ in $\pi_{i+1}$, $b \in \Sigma$, add $(r, s)$ to LIST iff $[r]$ and $[s]$ are distinct
    $b$-successors to $[q]$.
    Increase $i$ by one.
end while
Termination: Let $f = i$ and output the acceptor $A_0/\pi_f$.
```

In Fig. 1, for example, if $h = i$ then S-UPDATE removes this nondeterminism by merging states 6 and 7. Note this merge does not alter the relevant character of the acceptor or the language recognized by the acceptor. If merging two states creates additional nondeterminism (e.g. $l = k$ in Fig. 1), then the $b$-successors of the source of non-determinism are added to LIST.

**Theorem 5.** *Let $S$ be any finite sample. Then $L(PT(S)/\pi_{final}) \in LRI$.*

*Proof.* Let $A = PT(S)/\pi_{final}$. Let $A'$ be the acceptor obtained by submitting $A$ to S-UPDATE which removes sources of nondeterminism by merging states. It is sufficient to show $L(A') \in LRI$ and $L(A') = L(A)$. The proof is by induction. The assumptions here are inductive hypothesis. Assume there is a partition of $\pi_i$ such that $A/\pi_i = (Q_i, I_i, F_i, \delta_i)$ where

1. there is only one final state $[q_f]_{\pi_i}$,
2. all loops in $A$ pass through $[q_f]/\pi_i$,

3. no state other than $[q_f]_{\pi_i}$ has more than one 1-leader,
4. $L(A/\pi_i) = L(A)$.
5. if there exist distinct $q_0, q_1, q_2 \in Q_i$, and $b \in \Sigma$ such that $[q_1]_{\pi_i}$ and $[q_2]_{\pi_i}$ are b-successors to $[q_0]_{\pi_i}$, then any path connecting $[q_1]_{\pi_i}$ to $[q_2]_{\pi_i}$ (or vice versa) goes through $[q_0]_{\pi_i}$

Note if the conditional in (5) is false and (1-5) holds, then $L(A/\pi_i)$ is deterministic and hence $A/\pi_i$ belongs to $LRI$.

If the conditional in (5) is true, we show the acceptor obtained by merging $[q_1]_{\pi_i}$ and $[q_2]_{\pi_i}$ eliminates this nondeterminism but maintains properties (1-4). Let $\pi_{i+1}$ obtain by merging $[q_1]\pi_i$ and $[q_2]_{\pi_i}$. Since $\pi_i$ has only one final state, $\pi_{i+1}$ must as well (1). If $[q_1]\pi_i \neq [q_f]\pi_i$ and $[q_2]\pi_i \neq [q_f]\pi_i$ then it follows that the 1-leaders of $[q_1]\pi_i$ is $\{b\}$, as it is for for $[q_2]_{\pi_i}$. Therefore $[q_1]\pi_{i+1} = [q_2]\pi_{i+1}$ also has one 1-leader, namely $\{b\}$. On the other hand, if either $[q_1]\pi_i$ or $[q_1]_{\pi_i}$ equals $[q_f]_{\pi_i}$ then the b-successors of $[q_f]_{\pi_{i+1}} = [q_f]_{\pi_i}$. In any situation, it follows that no state other than $[q_f]_{\pi_{i+1}}$ has more than one 1-leader (3). Thus merging $[q_1]/\pi_i$ and $[q_2]/\pi_i$ creates no loops that do not go through $[q_f]/\pi_{i+1}$. It follows that all loops in $A/\pi_{i+1}$ pass through $[q_f]/\pi_{i+1}$ (2). Finally it follows that $L(A/\pi_{i+1}) = L(A/\pi_i)$ since any path connecting $[q_1]_{\pi_i}$ and $[q_2]_{\pi_i}$ (or vice versa) goes through $[q_0]_{\pi_i}$ (4). Thus this merging eliminates the nondeterminism at $[q_0]_{\pi_i}$ but maintains properties (1-4).

The base of the induction is established with $\pi_0$ the trivial partition of $A$. Since only final states are merged in $PT(S)$, $A$ contains only 1 final state (1). Also, since $PT(S)$ is acyclic, all loops in $A$ pass through $q_f$ as any loops are the result of merging states (Remark 1) (2). No state other than $q_f$ has more than one 1-leader (Remark 2) (3). Also, clearly $L(A/\pi_0) = L(A)$ (4). This completes the induction and so the final partition obtained by S-UPDATE $\pi_f$ is such that $L(A) = L(A/\pi_f) \in LRI$. □

Supplementing $PT(S)/\pi_{final}$ with Algorithm 1 provides an algorithm almost identical to the algorithm ZR in [5]. In ZR, pairs of states are placed on LIST if they are to be merged. LIST is initialized to include all final states, and pairs of states are added if they are a source of non-forward-determinism or non-reverse-determinism. Adding S-UPDATE to the computation of $PT(S)/\pi_{final}$ is exactly the same except there is no procedure for updating LIST when two blocks share the same $b$-predecessors (i.e. reverse-determinism is not enforced). Because it does strictly less than ZR, which is tractable, supplementing $PT(S)/\pi_{final}$ with Algorithm 1 is also tractable.

The lemma and theorems below establish that a learner which computes $PT(S)/\pi_{final}$ at each point in the text identifies $LRI$ in the limit.

**Lemma 2.** *Let $S$ be any nonempty positive sample, $PT(S)$ the prefix tree for $S$, and $\pi_f$ the final partition found by applying Algorithm 1 to $PT(S)/\pi_{final}$. Then $\pi_f$ is the finest partition $\pi$ such that $(PT(S)/\pi_{final})/\pi_f$ is LRI.*

*Proof.* The proof (by induction) is essentially identical to the second part of Lemma 25 in [5]. □

**Theorem 6.** *Let $S$ be any nonempty finite sample. Then $L(PT(S)/\pi_{final})$ is the smallest language in LRI which contains $S$.*

*Proof.* Theorem 5 establishes that $L(PT(S)/\pi_{final}) \in LRI$. Let $L$ be any LRI language containing $S$ and let $\pi$ be the restriction of the partition $\pi_L$to the elements of $Pref(S)$. Lemma 1 shows that $PT(S)/\pi$ is isomorphic to a subacceptor of $A_T(L)$, and it follows that $L(PT(S)/\pi)$ is contained in $L$. Theorem 3 shows that $A_T(L)$ is LRI, and thus $PT(S)/\pi$ is LRI, by Remark 3. By Lemma 2, $\pi_f$ therefore refines $\pi$. Hence, $L(PT(S)/\pi)$ is contained in $L$, and $L(PT(S)/\pi_{final})$ is the smallest LRI language containing S. □

**Theorem 7.** *LRI is identifiable in the limit from positive data.*

*Proof.* Let $\phi(t) = PT(S_t)/\pi_{final}$. By Theorem 4, $L$ contains a characteristic sample $S_0$. For any text $T$ for $L \in LRI$, there is a $i$ such that $S_0 \subseteq range(T_i)$. For $n \geq i$, $L(PT(T_i)/\pi_f)$ is the smallest $LRI$ language containing $T_i$ by Theorems 5 and 6. Since $T_i$ contains characteristic $S_0$, this is $L$. Thus $\phi$ converges to $L$. □

## 6 Right-to-Left Iterative Languages

RLI languages are defined as the intersection of the one-start-reverse deterministic languages ($\mathcal{L}_{1srd}$) and the reverse of $\mathcal{L}_{LL^*fin}$:

1. $\mathcal{L}_{1srd} = \{L : \text{whenever} u, v \in L, H_L(u) = H_L(v)\}$.
2. $\mathcal{L}_{L^*Lfin} = \{L_1^* \cdot L_2 : L_1, L_2 \in \mathcal{L}_{fin}\}$

*RLI* languages are the reverse of languages in *LRI*. *RLI* languages are exactly the ones recognizable by acceptors which are reverse deterministic, have at most one start state, and whose loops, if there are any, pass through the start state, if there is one. A schematic is obtained by reversing the acceptor in Fig. 1. *RLI* is is identifiable in the limit from a process which essentially merges initial states in suffix trees. The characteristic sample for language $L_1^* \cdot L_2 \in RLI$ is $L_1 \cdot L_2 \cup L_2$. The theorems establishing these results parallel exactly those of §5. Simply subsitute $ST(S)$, $start$, $A_H(L)$, $b$-predecessors, P-UPDATE and so on for $PT(S)$, $final$, $A_T(L)$, $b$-successors, and S-UPDATE, respectively. (P-UPDATE eliminates reverse determinisim by merging states with the same b-predecessors.)

Finally, [10] introduces a family of function-distinguishable language classes, of which ZR is one such class. It would be interesting to relate the results here to the ones obtained in that work.

## 7 Conclusion

LRI and RLI languages are previously unnoticed language classes which are infinite in size, identifiable in the limit from positive data, closely related to the zero-reversible languages, and relevant to a hypothesis regarding a universal property of phonotactic patterns. Understanding these classes not only begins to shed light on the neighborhood-distinct hypothesis, but also on the algebra underlying the state-merging operations, the reverse operator, prefix and suffix trees, and tail and head canonical acceptors. I hope this algebra is soon made clear, that the question marks in Table 1 are soon filled, and that future research investigates complex functions, like the neighborhood function, which are defined compositionally in terms of simpler functions. Finally, the day is not far off for three communities with overlapping interests to come together to develop a successful research program: the grammatical inference community which understands how different kinds of logically possible patterns could be learned, linguists who are familiar with natural language patterns, and acquisitionists who are experimenting with models of how children acquire grammar.

## References

1. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
2. Marcus, G.: Negative evidence in language acquisition. Cognition 46, 53–85 (1993)
3. Angluin, D.: Finding patterns common to a set of strings. Journal of Computer and System Sciences 21, 46–62 (1980)
4. Angluin, D.: Inductive inference of formal languages from positive data. Information Control 45, 117–135 (1980)
5. Angluin, D.: Inference of reversible languages. Journal for the Association of Computing Machinery 29(3), 741–765 (1982)
6. Garcia, P., Vidal, E., Oncina, J.: Learning locally testable languages in the strict sense. In: Proceedings of the Workshop on Algorithmic Learning Theory, pp. 325–338 (1990)
7. Muggleton, S.: Inductive Acquisition of Expert Knowledge. Addison-Wesley, Reading (1990)
8. Kanazawa, M.: Identification in the limit of categorical grammars. Journal of Logic, Language, and Information 5, 115–155 (1996)
9. Denis, F., Lemay, A., Terlutte, A.: Some classes of regular languages identifiable in the limit from positive data. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 63–76. Springer, Heidelberg (2002)
10. Fernau, H.: Identification of function distinguishable languages. Theoretical Computer Science 290, 1679–1711 (2003)
11. Yokomori, T.: Polynomial-time identification of very simple grammars from positive data. Theoretical Computer Science 298(1), 179–206 (2003)
12. Oates, T., Armstrong, T., Bonache, L.B.: Inferring grammars for mildly context-sensitive languages in polynomial-time. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 137–147. Springer, Heidelberg (2006)
13. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
14. Greenberg, J.: Some universals of grammar with particular reference to the order of meaningful elements. In: Universals of Language, pp. 73–113. MIT Press, Cambridge (1963)
15. Mairal, R., Gil, J. (eds.): Linguistic Universals. Cambridge University Press, Cambridge (2006)

16. Stabler, E.P.: Computational models of language universals: Expressiveness, learnability and consequences. In: Cornell Symposium on Language Universals (2007)
17. Shieber, S.: Evidence against the context-freeness of natural language. Linguistics and Philosophy 8, 333–343 (1985)
18. Kobele, G.: Generating Copies: An Investigation into Structural Identity in Language and Grammar. PhD thesis, University of California, Los Angeles (2006)
19. Johnson, C.D.: Formal Aspects of Phonological Description. Mouton, The Hague (1972)
20. Kaplan, R., Kay, M.: Regular models of phonological rule systems. Computational Linguistics 20(3), 331–378 (1994)
21. Heinz, J.: The Inductive Learning of Phonotactic Patterns. PhD thesis, University of California, Los Angeles (2007)
22. Pullum, G., Rogers, J.: Aural pattern recognition experiments and the subregular hierarchy. In: Kracht, M. (ed.) Proceedings of 10th Mathematics of Language Conference, pp. 1–7. University of California, Los Angeles (2007)
23. Jusczyk, P., Cutler, A., Redanz, N.: Infants' preference for the predominant stress patterns of english words. Child Development 64, 675–687 (1993)
24. Jusczyk, P., Luce, P., Charles-Luce, J.: Infants' sensitivity to phonotactic patterns in the native language. Journal of Memory and Language 33, 630–645 (1994)
25. Mattys, S., Jusczyk, P.: Phonotactic cues for segmentation of fluent speech by infants. Cognition 78, 91–121 (2001)
26. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge (1971)
27. Kracht, M.: The Mathematics of Language. Mouton de Gruyter, Berlin (2003)
28. Hansson, G.: Theoretical and typological issues in consonant harmony. PhD thesis, University of California, Berkeley (2001)
29. Hayes, B.: Metrical Stress Theory. Chicago University Press (1995)
30. Heinz, J.: Learning quantity insensitive stress systems via local inference. In: Proceedings of the Eighth Meeting of the ACL Special Interest Group in Computational Phonology at HLT-NAACL, New York City, USA, pp. 21–30 (2006)
31. Heinz, J.: On the role of locality in learning stress patterns. Phonology (to appear)
32. Jain, S., Osherson, D., Royer, J.S., Sharma, A.: Systems That Learn: An Introduction to Learning Theory, 2nd edn. The MIT Press, Cambridge (1999)
33. Case, J., Moelius, S.: Parallelism increases iterative learning power. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 49–63. Springer, Heidelberg (2007)
34. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (2001)
35. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27, 125–138 (1997)

# Learning Bounded Unions of Noetherian Closed Set Systems Via Characteristic Sets

Yuuichi Kameda[1], Hiroo Tokunaga[1], and Akihiro Yamamoto[2]

[1] Department of Mathematics and Information Sciences
Graduate School of Science and Engineering,
Tokyo Metropolitan University
1-1 Minami-Ohsawa, Hachioji-shi 192-0397 Japan
tokunaga@tmu.ac.jp, kameda-yuuiti@ed.tmu.ac.jp
[2] Graduate School of Informatics, Kyoto University
Yoshida Honmachi, Sakyo-ku, Kyoto 606-850 Japan
akihiro@i.kyoto-u.ac.jp

**Abstract.** In this paper, we study a learning procedure from positive data for bounded unions of certain class of languages. Our key tools are the notion of characteristic sets and hypergraphs. We generate hypergraphs from given positive data and exploit them in order to find characteristic sets.

## 1 Introduction

In this paper, we study a learning procedure from positive data for a certain class of languages. In the following, “learning” always means “learning from positive data.” The class of languages we consider is so called a *closed set system* (see §2 for its definition). In [4], we studied inferability of a closed set system in order to understand relations between the class of ideals of the polynomial ring and inferability from positive data. The polynomial ring is a fundamental object in algebra, and Hilbert’s basis theorem about the finite generation of its ideals has been historically important. In [5], Hayashi pointed out that if we consider ideals of the polynomial ring as formal languages, the statement of Hilbert’s basis theorem can be understood as inferability from positive data. In fact, Stephan and Ventsov [9] showed that a finite basis of any ideal of a commutative ring is regarded as a finite tell-tale. In [4], we introduced a notion of a *Noetherian closed set system* and proved that a closed set system $\mathcal{L}$ has finite elasticity if and only if it is Noetherian. Hence by the result of Wright et al. (Theorem 2.4 in §2), the class of bounded union $\cup^{\leq k}\mathcal{L}$ also has finite elasticity. From the proof of Theorem 2.4, however, we do not know how its learning procedure looks like. On the other hand, Kobayashi introduced the notion of a *characteristic set* in [7] and proved that $(i)$ if the class $\mathcal{L}$ has finite elasticity, then every language in $\mathcal{L}$ has a characteristic set and $(ii)$ $\mathcal{L}$ is inferable from positive data if every language in $\mathcal{L}$ has a characteristic set. Our goal of this note is to give a learning procedure for bounded unions of certain class of Noetherian closed set systems by using characteristic sets.

The contents of this paper is as follows. In §2, we summarize some facts on inferablity from positive data and closed set systems. In §3, we give a learning procedure of bounded unions of a Noetherian closed set system under certain settings as follows:

($i$) Given a closed set $L \in \mathcal{L}$ and its characteristic set $F$, there exists an algorithm to compute a characteristic set of $L$ in $\cup^{\leq k}\mathcal{L}$ from $F$.
($ii$) $\cup^{\leq k}\mathcal{L}$ is compact (See §2 for its definition).

Our procedure is given by generating a certain hypergraph, which is main feature of our result. In §§4 and 5, we apply our procedure to bounded unions of the class of ideals of the polynomial ring and tree pattern languages, respectively.

## 2 Preliminaries

### 2.1 Inferability from Positive Data

In this article, a *language* $L$ is a subset of some countable set $U$ such that $L$ is expressed $L(G)$ by some finite expression $G$. We call this finite expression a *hypothesis*. A set of all hypotheses $\mathcal{H}$ is called a *hypothesis space*. Let $\mathcal{L}$ be the set of all languages $\{L(G) \mid G \in \mathcal{H}\}$. We assume that $\mathcal{L}$ is *uniformly recursive*, that is, there is a recursive function $f(w, G)$ such that $f(w, G) = 1$ iff $w \in L(G)$ for every $w \in U$ and $G \in \mathcal{H}$.

A *positive data* (or *positive presentation*) of $L \in \mathcal{L}$ is an infinite sequence $\sigma : s_1, s_2, \ldots$ of elements of $L$ such that $L = \{s_1, s_2, \ldots\}$. An *inference algorithm* $M$ is that:

- $M$ receives incrementally an elements of a positive data $\sigma$ of a language,
- $M$ outputs a hypothesis $G_n \in \mathcal{H}$ when $M$ receives $n$-th element of $\sigma$.

$\mathcal{L}$ is *inferable in the limit from positive data* if there exists an inference algorithm $M$ satisfies that for all $L \in \mathcal{L}$ and an arbitrary positive data of $L$, the output sequence of $M$ converges to a hypothesis $G$ such that $L(G) = L$.

A *finite tell-tale* of $L \in \mathcal{L}$ is a finite subset $S$ of $L$ such that $L$ is a minimal in the class $\{L' \in \mathcal{L} \mid S \subset L'\}$ with respect to set inclusion. If $L$ is minimum, $S$ is called a *characteristic set* of $L$. Note that the idea of characteristic set is essentially the same as that of *test set* in [6].

**Theorem 2.1.** ([1]) *$\mathcal{L}$ is inferable in the limit from positive data if and only if there exists a procedure to enumerate elements of a finite tell-tale of every $L \in \mathcal{L}$.*

**Theorem 2.2.** ([7]) *If every $L \in \mathcal{L}$ has a characteristic set, then $\mathcal{L}$ is inferable from positive data.*

We say that $(i)$ $\mathcal{L}$ has *finite thickness* if the set $\{L \in \mathcal{L} \mid w \in L\}$ is finite for any $w \in U$ and $(ii)$ $\mathcal{L}$ has *infinite elasticity* if there exists an infinite sequence $w_0, w_1, \ldots$ of elements of $U$ and infinite sequence $L_1, L_2, \ldots$ of languages such that $\{w_0, \ldots, w_{n-1}\} \subset L_n$ but $w_n \notin L_n$. We say that $\mathcal{L}$ has *finite elasticity* if it does not have infinite elasticity.

**Theorem 2.3.** ([7],[8]) (1) *If $\mathcal{L}$ has finite elasticity, then every $L$ in $\mathcal{L}$ has a characteristic set.*
(2) *If $\mathcal{L}$ has finite thickness, then $\mathcal{L}$ has finite elasticity.*

We define a class of the union of languages as follows:

$$\mathcal{L} \cup \mathcal{L}' = \{L_1 \cup L_2 \mid L_1 \in \mathcal{L}, L_2 \in \mathcal{L}'\},$$
$$\cup^{\leq k}\mathcal{L} = \{L_1 \cup \ldots \cup L_m \mid m \leq k, L_i \in \mathcal{L}\ (i = 1, \ldots, m)\}.$$

It is known that

**Theorem 2.4.** ([12]) *If $\mathcal{L}$ and $\mathcal{L}'$ have finite elasticity, then $\mathcal{L} \cup \mathcal{L}'$ has finite elasticity.*

It immediately follows that, if $\mathcal{L}$ has finite elasticity, then $\cup^{\leq k}\mathcal{L}$ also has. Therefore, by Theorems 2.2 and 2.3, we have:

**Corollary 2.1.** *If $\mathcal{L}$ has finite elasticity, then $\cup^{\leq k}\mathcal{L}$ is inferable from positive data.*

**Definition 2.1.** *$\cup^{\leq k}\mathcal{L}$ is said to be compact if it satisfies the following condition:*
*For each $m \leq k$ and $L, L_i \in \mathcal{L}$ $(i = 1, \ldots, m)$, if $L \subset L_1 \cup \ldots \cup L_m$, then there exists $i_0$ such that $L \subset L_{i_0}$.*

## 2.2 Closed Set System

Let $2^U$ be the power set of $U$. A mapping $C : 2^U \to 2^U$ is called a *closure operator* if $C$ satisfies:
(CO1) $X \subset C(X)$,
(CO2) $C(C(X)) = C(X)$, and
(CO3) $X \subset Y \Rightarrow C(X) \subset C(Y)$,
where $X$ and $Y$ are arbitrary subsets of $U$. A set $X \subset U$ is called *closed* if $X = C(X)$. A *closed set system* $\mathcal{C}$ is the class of all closed sets of a closure operator.

**Remark 2.1.** In a closed set system, the intersection of arbitrary number of closed sets is closed, but the union of closed sets is not necessarily closed.

In the following, we regard $\mathcal{C}$ as a class of languages and assume that it is recursive. If a closed set $X \in \mathcal{C}$ is represented $X = C(Y)$ for some finite set $Y \subset U$, $X$ is called a *finitely generated closed set.*

**Lemma 2.1.** ([4]) *Let $X = C(Y)$ be a closed set. The followings are equivalent:*
*1. $Y$ is finite,*
*2. $Y$ is a finite tell-tale of $X$, and*
*3. $Y$ is a characteristic set of $X$.*

An immediate consequence of Lemma 2.1 and Theorem 2.1 is as follows:

**Corollary 2.2.** *$\mathcal{C}$ is inferable from positive data if and only if every closed set is finitely generated.*

A closed set system $\mathcal{C}$ is *Noetherian* if it contains no infinite strictly ascending chain of closed sets. This condition is equivalent to finite elasticity [4, Theorem 7]. Hence it follows that:

**Corollary 2.3.** *A Noetherian closed set system is inferable from positive data.*

## 3 Main Result

As for inferability of closed set systems from positive data, we refer to [4], and use result there freely.

Let $\mathcal{L}$ be a Noetherian closed set systems over some set $U$ and $C$ denote its closure operator. By [4, Theorem 7], $\mathcal{L}$ has finite elasticity and it implies that the class $\cup^{\leq k}\mathcal{L}$ also has finite elasticity. In particular, by [4, §3], any element $L$ of $\mathcal{L}$ is of the form $L = C(F)$ for a finite subset $F$ of $U$. In this section, we consider learning procedure for $\cup^{\leq k}\mathcal{L}$.

**Remark 3.1.** For an element $L_1 \cup \ldots \cup L_m \in \cup^{\leq k}\mathcal{L}$, we assume that $L_i \not\subseteq L_j$ for any $i, j (i \neq j)$.

Let $F$ be a finite subset of $U$. $F$ is a characteristic set of $C(F)$ in $\mathcal{L}$. Since $C(F)$ is also a member of $\cup^{\leq k}\mathcal{L}$, $C(F)$ has a characteristic set in $\cup^{\leq k}\mathcal{L}$, and we denote it by $\chi(C(F), \cup^{\leq k}\mathcal{L})$ (we may assume that $F \subseteq \chi(C(F), \cup^{\leq k}\mathcal{L})$). Throughout this section, we assume the following:

($*$) There exists an algorithm to compute $\chi(C(F), \cup^{\leq k}\mathcal{L})$ from $F$.

In §§4 and 5, we give examples of Noetherian closed set systems satisfying ($*$).

Let $L_1 \cup \ldots \cup L_m \in \cup^{\leq k}\mathcal{L}$ and let $\sigma : f_1, f_2, \ldots, f_n, \ldots$ be a positive data of $L_1 \cup \ldots \cup L_m$. We inductively define a hypergraph denoted by $\mathcal{G}_n$ having the set of vertices $V(\mathcal{G}_n) = \{f_1, \ldots, f_n\}$ as follows:

**Inductive definition of $\mathcal{G}_n$**
Let $V(\bullet)$ and $HE(\bullet)$ denote the set of vertices and hyperedges of a hypergraph $\bullet$, respectively.

For $n = 1$, we put

$$V(\mathcal{G}_1) = \{f_1\}, \quad HE(\mathcal{G}_1) = \{\{f_1\}\}.$$

Suppose that $\mathcal{G}_n$ is already given and $f_{n+1}$ is presented. We construct $\mathcal{G}_{n+1}$ in the following way:

```
Procedure 1: Construction of $\mathcal{G}_{n+1}$ from $\mathcal{G}_n$;
Input: $f_{n+1}$ and $\mathcal{G}_n$;
Output: a hypergraph $\mathcal{G}_{n+1}$;
begin
1.  put $V = V(\mathcal{G}_n) \cup \{f_{n+1}\}$ and $HE = HE(\mathcal{G}_n)$;
2.  for each subset $F \subset V$ such that $f_{n+1} \in F$ and $\sharp(F) \geq 2$ do begin
3.      let $E = \chi(C(F), \cup^{\leq k}\mathcal{L})$;
4.      if $E \subset V$ then begin
5.          for each element $\mathcal{E}$ of $HE$ do
6.              if $\mathcal{E} \subset E$ then remove $\mathcal{E}$ from $HE$;
7.          add $E$ to $HE$;
8.      end;
9.  end;
10. if there is no $\mathcal{E} \in HE$ such that $f_{n+1} \in \mathcal{E}$ then add $\{f_{n+1}\}$ to $HE$;
11. return $\mathcal{G}_{n+1} = (V, HE)$;
end.
```

Note that $\{f_{n+1}\}$ is a characteristic set of $C(\{f_{n+1}\})$ in $\cup^{\leq k}\mathcal{L}$.
We are now in a position to give our learning procedure:

```
Procedure 2: Learning $\cup^{\leq k}\mathcal{L}$;
Input: a positive presentation $\sigma : f_1, f_2, \ldots, f_n, \ldots$ for $L_1 \cup \ldots \cup L_m$;
Output: a sequence of at most $k$-tuples of characteristic sets
        $(\chi_1^{(1)}, \ldots, \chi_{m_1}^{(1)}), (\chi_1^{(2)}, \ldots, \chi_{m_2}^{(2)}), \ldots$ ;
begin
1.  $S = \emptyset$; /*Possible candidates for characteristic sets*/
2.  Put $n = 1$;
3.  repeat
4.      construct the hypergraph $\mathcal{G}_n$ for $f_1, f_2, \ldots, f_n$;
5.      put $S = HE(\mathcal{G}_n)$;
6.      choose at most $k$ maximal elements from $S$ with respect to the
        order as below;
7.      output (at most) $k$-tuple in 6;
8.      add 1 to $n$;
9.  forever;
end.
```

We define an ordering on $S$ as follows:

$\chi_1 < \chi_2 \Leftrightarrow C(\chi_1) \subsetneq C(\chi_2)$
    `ELSE` $C(\chi_1) = C(\chi_2)$ and $\chi_1 \prec \chi_2$ under a certain suitable ordering $\prec$.

The ordering $\prec$ does not affect the validity of **Procedure 2**, so we can adopt a convenient ordering (for example, the order of appearance in $S$.)

**Remark 3.2.** Note that $C(\chi_i^{(n)}) \not\subseteq C(\chi_j^{(n)})$ for any $i, j$ $(i \neq j)$.

Now our theorem is the following:

**Theorem 3.1.** *Suppose that $\cup^{\leq k}\mathcal{L}$ is compact. $\cup^{\leq k}\mathcal{L}$ is identifiable in the limit from positive data via **Procedure 2**.*

We need some lemmas to prove Theorem 3.1.

**Lemma 3.1.** *Let $\mathcal{E}$ be an arbitrary hyperedge of $\mathcal{G}_n$. Then $C(\mathcal{E}) \subset L_1 \cup \ldots \cup L_m$. Moreover, if $\mathcal{L}$ is compact, then there exists $L_i$ such that $C(\mathcal{E}) \subseteq L_i$.*

*Proof.* By our construction of $\mathcal{G}_n$, there exists $F \subseteq V(\mathcal{G}_n)$ such that $\mathcal{E} = \chi(C(F), \cup^{\leq k}\mathcal{L})$. Since $\mathcal{E} \subset C(F)$, $C(\mathcal{E}) \subseteq C(C(F)) = C(F)$. On the other hand, $\mathcal{E}$ is also a characteristic set of $C(F)$ in $\mathcal{L}$, $C(\mathcal{E}) \supseteq C(F)$, i.e., $C(\mathcal{E}) = C(F)$. Moreover, since $\mathcal{E}$ is a characteristic set of $C(F)$ in $\mathcal{L}^{\leq k}$, $C(F) \subseteq L_1 \cup \ldots \cup L_m$. The second statement is immediate from the definition of compactness.

**Lemma 3.2.** *Suppose that $\cup^{\leq k}\mathcal{L}$ is compact.* (1) *Let $L_1, \ldots, L_m$ be distinct members of $\mathcal{L}$. If $L_i \not\subset L_j$ for all $i, j (i \neq j)$, then $L_i \not\subset \cup_{j=1, j\neq i}^{m} L_j$.* (2) *Let $M \in \mathcal{L}$ and let $L_1, \ldots, L_m$ be as above. If $M \subseteq L_1 \cup \ldots \cup L_m$ and $L_i \subseteq M$ for some $i$, then $L_i = M$.*

*Proof.* (1) If $L_i \subset \cup_{j=1, j\neq i}^{m} L_j$, then $L_i \subseteq L_{j_0}$ for some $j_0$ by compactness. This contradicts to our assumption. (2) By compactness, there exists $L_{j_0}$ such that $M \subseteq L_{j_0}$. Hence $L_i \subseteq M \subseteq L_{j_0}$. By our assumption, $L_i = L_{j_0}$.

**Proof of Theorem 3.1.** Let $L_1 \cup \ldots \cup L_m$ be an arbitrary element in $\cup^{\leq k}\mathcal{L}$. Suppose that $L_i = C(F_i)$, where $F_i$ is a finite subset of $L_i$. Since $F_i$ can be considered as $\chi(L_i, \mathcal{L})$, at a certain finite step $N_0$, all elements of $F_i$ are presented. Therefore, at a certain step $N$ after the step $N_0$, one can assume that all elements of $\chi(C(F_i), \cup^{\leq k}\mathcal{L})$ $for$ $i = 1, \ldots, m$ are presented. Let $\mathcal{E}_{1,N}, \ldots, \mathcal{E}_{m_N,N}$ be hyperedges of $\mathcal{G}_N$ as in Output of Procedure 2. By construction, each $\mathcal{E}_{j,N}$ is a characteristic set of closed set $C(\mathcal{E}_{j,N})$ contained in $L_1 \cup \ldots \cup L_m$. Note that $C(\mathcal{E}_{i,N}) \not\subseteq C(\mathcal{E}_{j,N})$ for any $i, j (i \neq j)$ by Remark 3.2.

**Claim.** For each $\chi(C(F_i), \cup^{\leq k}\mathcal{L})$, there exists a unique $\mathcal{E}_{j_i,N}$ of $HE(\mathcal{G}_N)$ with $C(F_i) = C(\mathcal{E}_{j_i,N})$.

By our construction of $\mathcal{G}_N$, $\chi(C(F_i), \cup^{\leq k}\mathcal{L})$ is either added as a hyperedge or contained in a hyperedge added at a certain step. Hence there exists a hyperedge $\mathcal{E}_i$ of $\mathcal{G}_N$ such that $\chi(C(F_i), \cup^{\leq k}\mathcal{L}) \subseteq \mathcal{E}_i$ for each $i$. Since $C(F_i) = C(\chi(C(F_i), \cup^{\leq k}\mathcal{L})) \subseteq C(\mathcal{E}_i)$, $C(\mathcal{E}_i) = C(F_i)$ by Lemmas 3.1 and 3.2(2). As $L_i = C(\mathcal{E}_i)$ is a maximal element of $\mathcal{L}$ contained in $L_1 \cup \ldots \cup L_m$, there exists a hyperedge $\mathcal{E}_{j,N}$ such that $L_i = C(F_i) = C(\mathcal{E}_{j,N})$ (Note that $\mathcal{E}_{j,N}$ is not necessarily equal to $\mathcal{E}_i$). Suppose that there exist two distinct $\mathcal{E}_{j_1,N}$ and $\mathcal{E}_{j_2,N}$ such that $\chi(C(F_i), \cup^{\leq k}\mathcal{L}) \subseteq \mathcal{E}_{j_l}$ $(l = 1, 2)$. Then $C(F_i) = C(\chi(C(F_i), \cup^{\leq k}\mathcal{L})) \subseteq C(\mathcal{E}_{j_l})$ $(l = 1, 2)$. By Lemma 3.2, $C(F_i) = C(\mathcal{E}_{j_1,N}) = C(\mathcal{E}_{j_2,N})$, but this contradicts to our assumption.

We finally show that $m_N = m$. By Claim, $m_N \geq m$. If $m_N > m$, then there exists $\mathcal{E}_{j_0,N}$ such that $(i)$ $C(\mathcal{E}_{j_0,N}) \neq L_i$ for $i = 1, \ldots, m$ and $(ii)$ $C(\mathcal{E}_{j_0,N}) \subset L_1 \cup \ldots \cup L_m$. But these condition implies that $C(\mathcal{E}_{j_0,N}) \subset C(\mathcal{E}_{j_i,N})$ for some $j_i$. This contradicts to our choice of $\mathcal{E}_{i,N}$ $(i = 1, \ldots, m_N)$.

**Remark 3.3.** Note that the hypotheses in our algorithm are not necessarily consistent. However, one can modify them into consistent ones without difficulty.

## 4 Learning Bounded Set Unions of Polynomial Ideals

We denote the set of polynomials of $n$ variables with $\mathbb{Q}$-coefficients by $\mathbb{Q}[x_1, \ldots, x_n]$. A subset $I$ of $\mathbb{Q}[x_1, \ldots, x_n]$ is called an *ideal* if it satisfies the following:

- For each $f, g \in I$, $f \pm g \in I$.
- For each $f \in I$ and $h \in \mathbb{Q}[x_1, \ldots, x_n]$, $hf \in I$.

We denote the set of all ideals by $\mathcal{I}$. For a finite subset $F = \{f_1, \ldots, f_r\} \subset \mathbb{Q}[x_1, \ldots, x_n]$, we define the ideal generated by $f_1, \ldots, f_r$, which is denoted by $\langle f_1, \ldots, f_r \rangle$ or $\langle F \rangle$, as follows:

$$\langle F \rangle := \{\sum_{i=1}^{r} h_i f_i \mid \ h_i \in \mathbb{Q}[x_1, \ldots, x_n]\}.$$

Note that the correspondence $F \mapsto \langle F \rangle$ defines a closure operator on $\mathbb{Q}[x_1, \ldots, x_n]$. By Hilbert's basis theorem for polynomial ideals, we have the following: for each $I \in \mathcal{I}$, there exists a finite set $F$ such that $I = \langle F \rangle$. An interpretation of this statement from machine learning view point is that "$\mathcal{I}$ has a finite elasticity." Hence, $\mathcal{I}$ is a Noetherian closed set system with the closure operator $F \mapsto \langle F \rangle$. Furthermore, the existence of a reduced Groebner basis for given $I$ in theory of Groebner basis says that one can take the set of reduced Groebner bases as a hypothesis space of $\mathcal{I}$.

The following lemma is a special case of [10, Theorem 9]. This lemma implies that $\cup^{\leq k}\mathcal{I}$ satisfies the condition $(*)$.

**Lemma 4.1.** *Let $I \in \mathcal{I}$. A characteristic set $\chi(I, \cup^{\leq k}\mathcal{I})$ can be constructed if the reduced Groebner basis $G = \{g_1, \ldots, g_r\}$ of $I$ is given.*

**Remark 4.1.** For instance we have an example of $\chi(I, \cup^{\leq k}\mathcal{I})$ as follows:

$$h_i = g_1 + c_i g_2 + \ldots + c_i^{r-1} g_r \ (i = 1, \ldots, M)$$

where $M = k(r-1)+1$ and $c_i$'s are distinct elements of $\mathbb{Q}$. Note that no $h_i$ will vanish since $\{g_1, \dots, g_r\}$ is the reduced Groebner basis.

**Remark 4.2.** This lemma also implies that $\cup^{\leq k}\mathcal{I}$ is compact: suppose that $I = \langle g_1, \dots, g_r \rangle$ is contained in $I_1 \cup \dots \cup I_m$. Let $M = m(r-1)+1$ and take $h_1, \dots, h_M$ as above. By the pigeon-hole principle, there exists some $j$ such that $I_j$ includes at least $r$ of $h_i$'s. This means $I \subset I_j$.

According to above arguments, we have:

**Theorem 4.1.** *$\cup^{\leq k}\mathcal{I}$ is identifiable in the limit from positive data via* ***Procedure 2****.*

**Example 4.1.** Let us consider learning $\langle x^2, y^3 \rangle \cup \langle x^3, y^2 \rangle \in \cup^{\leq 2}\mathcal{I}$. Let a positive presentation $\sigma$ be $x^2, y^3, y^2, x^2+y^3, x^3, x^3+y^2, \dots$ . By the argument of §3 of [10], we can take a characteristic set $\chi(\langle f, g \rangle, \cup^{\leq 2}\mathcal{I}) = \{f, g, f+g\}$ for distinct polynomials $f$ and $g$. The hyperedges of hypergraphs constructed by **Procedure 1** are as follows:

$$\begin{aligned}
HE_1 &= \{\{x^2\}\}, \\
HE_2 &= \{\{x^2\}, \{y^3\}\}, \\
HE_3 &= \{\{x^2\}, \{y^3\}, \{y^2\}\}, \\
HE_4 &= \{\{x^2, y^3, x^2+y^3\}, \{y^2\}\}, \\
HE_5 &= \{\{x^2, y^3, x^2+y^3\}, \{y^2\}, \{x^3\}\}, \\
HE_6 &= \{\{x^2, y^3, x^2+y^3\}, \{y^2, x^3, x^3+y^2\}\}.
\end{aligned}$$

Hence **Procedure 2** learns $\langle x^2, y^3 \rangle \cup \langle x^3, y^2 \rangle$ when $n = 6$.

## 5 Learning Bounded Unions of Tree Pattern Languages

In [3], Arimura et al. studied learnability of bounded union of tree pattern languages. However, they did not seem to use characteristic sets explicitly. We here give a procedure learning bounded unions of tree pattern languages by using our result in §3. Let $\Sigma$ be a finite set and $V$ be a countable set disjoint from $\Sigma$. The elements of $\Sigma$ and $V$ are called *symbols* and *variables*, respectively. We assume that there is a mapping *rank* that maps an element of $\Sigma$ to a non-negative integers. We define the rank of elements of $V$ to be zero. A *tree pattern* $p$ over $\Sigma$ is a tree satisfying following properties:

• $p$ has the root.
• $p$ is directed.
• $p$ is ordered.
• Each node of $p$ is labeled by elements of $\Sigma \cup V$.
• The number of children of each node is equal to the rank of the label of the node.

A *tree* over $\Sigma$ is a tree pattern over $\Sigma$ that has no nodes labeled by an element of $V$. $\mathcal{TP}_\Sigma$ and $\mathcal{T}_\Sigma$ denote the set of all tree patterns and all trees over $\Sigma$, respectively.

A *substitution* is a mapping $\theta$ from $V$ to $\mathcal{TP}_\Sigma$. $p\theta$ denotes the tree pattern obtained from applying a substitution $\theta$ to $p$. We define a relation on $\mathcal{TP}_\Sigma$ as follows: $p \preceq q \Leftrightarrow$ there exists a substitution $\theta$ such that $p = q\theta$. We denote $p \equiv q$ if $p \preceq q$ and $q \preceq p$, and call that $p$ and $q$ are equivalent. Note that $p \equiv q$ if and only if $p = q\theta$ for some renaming $\theta$ of variables.

**Lemma 5.1.** (1) *If $p \preceq q$ and $q \preceq r$, then $p \preceq r$.*
(2) *Let $|p|$ be the number of nodes of $p$. If $p \preceq q$, then $|p| \geq |q|$.*
(3) *For any $p \in \mathcal{TP}_\Sigma$, there are finitely many $q \in \mathcal{TP}_\Sigma$ such that $p \preceq q$.*

**Lemma 5.2.** *For any subset $S \neq \emptyset$ of $\mathcal{TP}_\Sigma$, there exists an element $lca(S)$ of $\mathcal{TP}_\Sigma$ such that:*

*(i) $p \preceq lca(S)$ for any $p \in S$,*
*(ii) if $p \preceq r$ for any $p \in S$, then $lca(S) \preceq r$.*

*$lca(S)$ is uniquely determined up to equivalence. It is called the* least common anti-instance *of $S$. If $S$ is finite, then $lca(S)$ can be computed in polynomial time [8].*

A *tree pattern language defined by $p$* is the set $L(p) = \{t \in \mathcal{T}_\Sigma \mid t \preceq p\}$. We denote the set of all tree pattern languages $\{L(p) \mid p \in \mathcal{TP}_\Sigma\}$ by $\mathcal{TPL}(\Sigma, V)$. We may omit $(\Sigma, V)$ if it is clear from the context.

**Lemma 5.3.** (1) *$p \preceq q \Leftrightarrow L(p) \subset L(q)$ for any $p, q \in \mathcal{TP}_\Sigma$.*
(2) *$L(t) = \{t\}$ for any $t \in \mathcal{T}_\Sigma$.*
(3) *$lca(t_1, t_2) \preceq p$ for any $p \in \mathcal{TP}_\Sigma$ and $t_1, t_2 \in L(p)$.*

In general, the class $\mathcal{TPL}$ itself may be not a closed set system. Hence we introduce a closed set system $\mathcal{C}$ over $\mathcal{TPL}$.
For $S \subset \mathcal{TP}_\Sigma$, we define $C(S) = \{p \in \mathcal{TP}_\Sigma \mid p \preceq lca(S)\}$. Note that $C(S) = C(lca(S))$.

**Lemma 5.4.** *$C$ is a closure operator on $\mathcal{TP}_\Sigma$.*

*Proof.* (CO1) Obvious by the definition of $lca$. (CO2) In general, $lca(C(S)) = lca(S)$ holds. Thus $C(C(S)) = C(lca(C(S))) = C(lca(S)) = C(S)$. (CO3) Suppose $S_1 \subset S_2 \subset \mathcal{TP}_\Sigma$. Clearly, $lca(S_1) \preceq lca(S_2)$. Lemma 5.1(1) implies $C(S_1) = C(lca(S_1)) \subset C(lca(S_2)) = C(S_2)$.

$\mathcal{C}$ denotes the closed set system defined by $C$. The following lemma indicates a fundamental relation between $\mathcal{TPL}$ and $\mathcal{C}$.

**Lemma 5.5.** *For every $p \in \mathcal{TP}_\Sigma$, $L(p) = C(p) \cap \mathcal{T}_\Sigma$.*

**Lemma 5.6.** (1) *$\mathcal{TPL}$ and $\mathcal{C}$ have finite elasticity.*
(2) *If $\sharp(\Sigma) > k$, $\cup^{\leq k}\mathcal{TPL}$ is compact.*
(3) *$\cup^{\leq k}\mathcal{C}$ is compact.*

*Proof.* (1) For any fixed $k \in \mathbb{N}$, the set $\{p \in \mathcal{TP}_\Sigma \mid |p| \leq k\}$ is finite up to equivalence. This fact and Lemma 5.1(2) imply that, for any $p \in \mathcal{TP}_\Sigma$, there are finitely many $q \in \mathcal{TP}_\Sigma$ such that $p \preceq q$. This means that $\mathcal{TPL}$ and $\mathcal{C}$ have finite thickness. Therefore, they have finite elasticity by Theorem 2.3.
(2) See [2].
(3) Suppose that $C(p) \subset C(p_1) \cup \ldots \cup C(p_m)$ $(m \leq k)$. Since $p \in C(p)$, there exists $i_0$ such that $p \in C(p_{i_0})$. Hence $C(p) \subset C(p_{i_0})$.

Let $\Sigma_0 = \{a \in \Sigma \mid rank(a) = 0\}$ and $\Sigma_+ = \{f \in \Sigma \mid rank(f) > 0\}$. In the following, we assume that neither $\Sigma_0$ nor $\Sigma_+$ is empty.

**Lemma 5.7.** (1) *For every $p \in \mathcal{TP}_\Sigma$, there exists a characteristic set $\chi(L(p), \mathcal{TPL})$ consisting of at most two elements.*
(2) *For every $S \subset \mathcal{TP}_\Sigma$, there exists a characteristic set $\chi(C(S), \mathcal{C})$ consisting of one element.*

**Lemma 5.8.** (1) *Suppose $\sharp(\Sigma_+) \geq k$. For every $p \in \mathcal{TP}_\Sigma$, there exists a characteristic set $\chi(L(p), \cup^{\leq k}\mathcal{TPL})$ consisting of at most $k+1$ elements. (In fact, there exists a set $\{t_1, \ldots, t_{k+1}\} \subset \mathcal{T}_\Sigma$ such that $lca(t_i, t_j) = p$ for each $i \neq j$. See [11] for detail.)*
(2) *For every $S \subset \mathcal{TP}_\Sigma$, there exists a characteristic set $\chi(C(S), \cup^{\leq k}\mathcal{C})$ consisting of one element.*

Lemma 5.8(2) makes algorithm learning $\cup^{\leq k}\mathcal{C}$ much simpler.

```
Procedure 3: Learning $\cup^{\leq k}\mathcal{C}$;
Input: a positive presentation $\sigma : q_1, q_2, \ldots, q_n, \ldots$ for
          $C(p_1) \cup \ldots \cup C(p_m)$;
Output: a sequence of at most $k$-tuples of tree patterns
          $(r_1^{(1)}, \ldots, r_{m_1}^{(1)}), (r_1^{(2)}, \ldots, r_{m_2}^{(2)}), \ldots$ ;
begin
1.   $S = \emptyset$; /*The set to memorize a given sequence of $q_1, \ldots, q_n$*/
2.   put $n = 1$;
3.   repeat
4.       add $q_n$ to $S$;
5.       choose at most $k$ maximal elements from $S$ with respect to $\preceq$
         up to equivalence;
6.       output (at most) $k$-tuple in 5;
7.       add 1 to $n$;
8.   forever
end.
```

We assume $\sharp(\Sigma_+) \geq k$ in order to make Lemma 5.8(1) holds. By using **Procedure 3**, $\cup^{\leq k}\mathcal{TPL}$ is inferred as follows:

```
Procedure 4: Learning $\cup^{\leq k}\mathcal{TPL}$;
Input: a positive presentation $\sigma : t_1, t_2, \ldots, t_n, \ldots$ for
        $L(p_1) \cup \ldots \cup L(p_m)$;
Output: a sequence of at most $k$-tuples of tree patterns
        $(q_1^{(1)}, \ldots, q_{m_1}^{(1)}), (q_1^{(2)}, \ldots, q_{m_2}^{(2)}), \ldots$ ;
begin
1.  generate "positive data" of $C(p_1) \cup \ldots \cup C(p_m)$ from $\sigma$;
2.  run Procedure 3 by "positive data" generated in 1;
3.  output the output of 2;
end.
Generation of "positive data" ($\mathcal{GPD}$);
4.  $S = \emptyset$; /*The set to memorize a given sequence of $t_1, \ldots, t_n$*/
5.  put $n = 1$;
6.  repeat
7.      add $t_n$ to $S$;
8.      output $t_n$;
9.      for each subset $F$ of $S$ with $t_n \in F$ and $\sharp(F) = k + 1$ do
10.         if $lca(t_i, t_j) = lca(F)$ for all $t_i, t_j \in F$ $(i \neq j)$ then
11.             output $lca(F)$;
12.     add 1 to $n$;
13. forever;
end.
```

**Theorem 5.1.** *$\cup^{\leq k}\mathcal{TPL}$ is identifiable in the limit from positive data via **Procedure 4**.*

*Proof.* It suffices to show that $\mathcal{GPD}$ generates a positive data for $A = C(p_1) \cup \ldots \cup C(p_m)$. Let $p$ be an arbitrary element of $A$. If $p \in \mathcal{T}_\Sigma$, then $p \in A \cap \mathcal{T}_\Sigma = L(p_1) \cup \ldots \cup L(p_m)$, so there exists a number $j$ such that $t_j = p$. Thus, $p$ is enumerated by step 8 of **Procedure 4**. If not, then there exists a set $F$ that satisfies the condition of step 11 by Lemma 5.8(1). Let $n_0$ be the least $n$ satisfying $\{t_1, \ldots, t_n\} \supset F$. It is clear that $p$ is enumerated at Step 11 when $n = n_0$.

We end this section by giving an example.

**Example 5.1.** Suppose $\Sigma = \{a, b, f, g\}, rank(a) = rank(b) = 0, rank(f) = 2, rank(g) = 1$, and $x, y \in V$. Let us consider learning $L(f(a, x)) \cup L(f(x, b)) \in \cup^{\leq 2}\mathcal{TPL}$. Let a positive presentation $\sigma$ be as follows:

$$t_1 = f(a, a),\ t_2 = f(a, f(a, b)),\ t_3 = f(b, b),$$
$$t_4 = f(a, g(a)),\ t_5 = f(a, b),\ t_6 = f(g(a), b), \ldots$$

This time **Procedure 4** learns $L(f(a, x)) \cup L(f(x, b))$ as follows:

•$n = 1 :$ $\mathcal{GPD}$ outputs $t_1$ and **Procedure 4** outputs $(t_1)$.
•$n = 2 :$ $\mathcal{GPD}$ outputs $t_2$ and **Procedure 4** outputs $(t_1, t_2)$.

•$n = 3$ : $lca(t_1, t_2) = f(a, x)$, $lca(t_1, t_3) = f(x, x)$, $lca(t_2, t_3) = f(x, y)$. Hence $\mathcal{GPD}$ outputs only $t_3$. **Procedure 4** chooses two larger elements from $\{t_1, t_2, t_3\}$ and output them.
•$n = 4$ : Since $lca(t_1, t_2) = lca(t_1, t_4) = lca(t_2, t_4) = f(a, x)$, $\mathcal{GPD}$ outputs $t_4$ and $f(a, x)$. **Procedure 4** outputs two maximal elements of $\{t_1, \ldots, t_4, f(a, x)\}$, that is, $f(a, x)$ and $t_3$.
•$n = 5$ : Since $lca(t_1, t_5) = lca(t_2, t_5) = lca(t_4, t_5) = f(a, x)$, $\mathcal{GPD}$ outputs $t_5$ and $f(a, x)$. **Procedure 4** outputs $f(a, x)$ and the larger element of $\{t_3, t_5\}$.
•$n = 6$ : Since $lca(t_3, t_5) = lca(t_3, t_6) = lca(t_5, t_6) = f(x, b)$, $\mathcal{GPD}$ outputs $t_6$ and $f(x, b)$. **Procedure 4** outputs $(f(a, x), f(x, b))$.

## 6 Conclusions

We have seen that the notion of characteristic set and its computability play important role to give a learning procedure of bounded unions of Noetherian closed set systems. The existence of characteristic set has not been used to give a concrete learning procedure. This is probably because the existence of a characteristic set for each language is weaker condition than finite thickness or finite elasticity. Also both finite thickness and finite elasticity are properties concerning family of language, while the existence of a characteristic set just depends on each language. The point of our paper is to put emphasis on a characteristic set and to show that it is useful for certain classes of languages.

## Acknowledgment

This research is partially supported by Grant-in-Aid 19300046 from JSPS. The second author is partially supported by K18-XI-234 from Kayamori Foundation of Informational Science Advancement.

## References

1. Angluin, D.: Inductive Inference of Formal Languages from Positive Data. Information and Control 45, 117–135 (1980)
2. Arimura, H., Shinohara, T., Otsuki, S.: Polynomial Time Inference of Unions of Tree Pattern, Languages. In: Proc. the 2nd Workshop on Algorithmic Learning Theory (ALT 1991), pp. 105–114 (1991)
3. Arimura, H., Shinohara, T., Otsuki, S.: A Polynomial Time Algorithm for Finding Finite Unions of Tree Patterns. In: Brewka, G., Jantke, K.P., Schmitt, P.H. (eds.) NIL 1991. LNCS, vol. 659, pp. 118–131. Springer, Heidelberg (1993)
4. de Brecht, M., Kobayashi, M., Tokunaga, H., Yamamoto, A.: Inferability of Closed Set Systems From Positive Data. In: Washio, T., Satoh, K., Takeda, H., Inokuchi, A. (eds.) JSAI 2006. LNCS (LNAI), vol. 4384, pp. 265–275. Springer, Heidelberg (2007)
5. Hayashi, S.: Mathematics Based on Learning. In: Cesa-Bianchi, N., Numao, M., Reischuk, R. (eds.) ALT 2002. LNCS (LNAI), vol. 2533, pp. 7–21. Springer, Heidelberg (2002)

6. Kapur, S., Bilardi, G.: On uniform learnability of language families. Information Processing Letters 44, 35–38 (1992)
7. Kobayashi, S.: Approximate Identification, Finite Elasticity and Lattice Structure of Hypothesis Space, Technical Report, CSIM 1996-2004, Dept. of Compt. Sci. and Inform. Math., Univ. of Electro-Communications (1996)
8. Lassez, J.L., Maher, M.J., Marriott, K.: Unification Revisited. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 587–626. Morgan-Kaufman, San Francisco (1988)
9. Stephan, F., Ventsov, Y.: Learning Algebraic Structures from Text. Theoretical Computer Science 268, 221–273 (2001)
10. Takamatsu, I., Kobayashi, M., Tokunaga, H., Yamamoto, A.: Computing Characteristic Sets of Bounded Unions of Polynomial Ideals. In: Satoh, K., Inokuchi, A., Nagao, K., Kawamura, T. (eds.) JSAI 2007. LNCS (LNAI), vol. 4914, pp. 318–329. Springer, Heidelberg (2008)
11. Tokunaga, H., Yamamoto, A.: Inductive Inference of Bounded Unions of Languages with Complete Characteristic Sets (in Japanese), SIG-FPAI-A703-01, 01-08 (2008)
12. Wright, K.: Identification of Unions of Languages Drawn from an Identifiable Class. In: Proc. of COLT 1989, pp. 328–388. Morgan-Kaufman, San Francisco (1989)

# A Learning Algorithm for Multi-dimensional Trees, or: Learning Beyond Context-Freeness

Anna Kasprzik

University of Trier

**Abstract.** We generalize a learning algorithm by Drewes and Högberg [1] for regular tree languages based on a learning model proposed by Angluin [2] to recognizable tree languages of arbitrarily many dimensions, so-called multi-dimensional trees. Trees over multi-dimensional tree domains have been defined by Rogers [3,4]. However, since the algorithm by Drewes and Högberg relies on classical finite state automata, these structures have to be represented in another form to make them a suitable input for the algorithm: We give a new representation for multi-dimensional trees which establishes them as a direct generalization of classical trees over a partitioned alphabet, and show that with this notation Drewes' and Högberg's algorithm is able to learn tree languages of arbitrarily many dimensions. Via the correspondence between trees and string languages ("yield operation") this is equivalent to the statement that this way even some string language classes beyond context-freeness have become learnable with respect to Angluin's learning model as well.

**Keywords:** MAT learning, multi-dimensional trees, finite-state.

## 1 Introduction

In the area of grammatical inference the problem of how to algorithmically infer (or "learn") a description of a formal language (e.g., a grammar or an automaton) on the basis of given examples or other information on that language is considered. Several learning models have been formulated, and based on those quite an amount of learning algorithms (mainly for regular languages or subclasses thereof) have been developed. In one of those models, proposed by Angluin [2] along with a P-time learning algorithm $L^*$ for regular string languages, the "learner" is helped by a "minimally adequate teacher" (MAT) who can answer two types of queries, namely if a given word is or is not a member of the language $U$ to be learned, and, for some finite-state automaton $\mathcal{A}$, if $\mathcal{A}$ correctly recognizes $U$. If not, the teacher will return a counterexample. The algorithm $L^*$ has been adapted by Sakakibara [5] to skeletal regular tree languages (regular sets of trees with unlabeled inner nodes) and then generalized by Drewes and Högberg [1] to regular tree languages throughout. As regular tree languages are a well-known generalization of regular string languages, this is a logical step.

We will generalize Drewes' and Högberg's algorithm even further to recognizable tree languages of arbitrarily many dimensions (sets of so-called *multi-dimensional trees*). Trees based on multi-dimensional tree domains have been defined by Rogers [3,4], along with finite-state automata for these trees. Labeled one-dimensional trees correspond to strings, and the automata recognizing them are equivalent to classical finite-state automata. The automata recognizing labeled two-dimensional trees are equivalent to classical finite-state tree automata.

Every multi-dimensional tree language has a set of strings – the *string yields* of its elements – associated with it which is obtained by reducing the number of dimensions of the trees step by step, down to the first, only retaining the outermost nodes and their connecting structure in each step (see [3] or Subsection 3.2 for a definition). As is well known, the sets of string yields of the languages recognized by (two-dimensional) finite-state tree automata coincide with the class of context-free languages. The sets obtained when reducing the number of dimensions of recognizable three-dimensional tree languages by one have an interesting linguistic aspect: They correspond exactly to the sets of trees generated by *(non-strict) Tree Adjoining Grammars* (see [3,4]), a special kind of tree formalism in which trees are built via *adjunction*, an operation which can be seen as a particular form of context-free tree rewriting. TAGs have been developed by Joshi [6] in connection with studies on the formal treatment of natural languages. Joshi [6] claimed the least class of formal languages containing all natural languages to be situated between the context-free and the context-sensitive languages in the Chomsky Hierarchy, and named it the class of *mildly context-sensitive languages*. The string sets associated with TAGs fulfil all necessary conditions for this class. TAGs are considered the standard model for mild context-sensitivity and are the foundation of a considerable amount of current work in applied computational linguistics. Rogers [4] conjectures that there might also be some linguistic phenomena that can best be handled via structures of more than three dimensions, and gives an amelioration of the standard TAG account of modifiers using four dimensions (see [3]).

The classes of sets of string yields associated with the recognizable multi-dimensional tree languages ordered by the number of dimensions form a (proper) infinite hierarchy properly contained in the context-sensitive class, with the classes of context-free languages (sets of string yields of two-dimensional tree sets) and the string languages associated with TAGs (sets of string yields of three-dimensional tree sets) as the first two steps. According to Rogers [3,4], this hierarchy coincides with Weir's Control Language Hierarchy [7].

It is a consequence of these correspondencies that by processing recognizable higher-dimensional descriptions of non-regular string languages instead of the string sets themselves, finite-state methods become applicable again (see [9,8]). Thus, just as by adapting Angluin's learning algorithm for regular string languages [2] to (skeletal) regular tree languages [5,1] context-free string languages have been made MAT-learnable, generalizing the adapted algorithm to recognizable tree languages of arbitrarily many dimensions and then recurring to the concept of yield can make even string language classes beyond context-freeness learnable under Angluin's MAT learning model as well.

As the learning algorithm by Drewes and Högberg [1] is based on classical finite-state tree automata and consequently on the concept of trees as terms over a partitioned alphabet, but Rogers' definition of multi-dimensional trees is based on tree domains, the algorithm cannot be used on these structures without representing them in another form. We will give a new term-like representation for multi-dimensional trees, which was introduced in [9] and which establishes them as a direct generalization of classical trees, along with an adapted definition of finite-state automata and of the yield operation, and show that this innovation enables Drewes' and Högberg's algorithm to learn languages of trees of arbitrarily many dimensions by proving that despite the modified input all its essential properties (including the ability to yield the desired output) stay preserved.

## 2 A Learning Algorithm for Regular Tree Languages

In this section, we are going to describe the learner for trees by Drewes and Högberg [1]. First of all, we need some basic notions about trees.

A *ranked alphabet* is a finite set of symbols, each associated with a rank $n \in \mathbb{N}$ (including 0). By $\Sigma_n$ we denote the set of all symbols in $\Sigma$ with rank $n$. Traditionally, every symbol is associated with a single rank only, but it is just as possible to admit several ranks for one symbol (see for example [5]), as long as there is a maximal admissible rank and the alphabet stays finite.

The set $T_\Sigma$ of all trees over $\Sigma$ is defined inductively as the smallest set of expressions such that $f[t_1, \ldots, t_n] \in T_\Sigma$ for every $f \in \Sigma_n$ and all $t_1, \ldots, t_n \in T_\Sigma$. $t_1, \ldots, t_n$ are the *direct subtrees* of the tree. The set $subtrees(t)$ consists of $t$ itself and all subtrees of its direct subtrees. Given a set $T$ of trees, $\Sigma(T)$ denotes the set of all trees of the form $f[t_1, \ldots, t_n]$ such that $f \in \Sigma_n$ for some $n$ and $t_1, \ldots, t_n \in T$. A subset of $T_\Sigma$ is called a tree language.

Let $\square$ be a special symbol of rank 0. A tree $c \in T_{\Sigma \cup \{\square\}}$ in which $\square$ occurs exactly once is a *context*, the set of all contexts over $\Sigma$ is denoted by $C_\Sigma$. For $c \in C_\Sigma$ and $s \in T_\Sigma$, $c[[s]]$ denotes the tree obtained by substituting $s$ for $\square$ in $c$. $depth(c)$ is the length of the path from the root to $\square$.

A *(total, deterministic) bottom-up finite-state tree automaton (fta)* $\mathcal{A} = (\Sigma, Q, \delta, F)$ has a ranked input alphabet $\Sigma$, finite state set $Q$, transition function $\delta$ assigning to every $f \in \Sigma_n$ and all $q_1, \ldots, q_n \in Q$ a state $\delta(q_1 \cdots q_n, f) \in Q$, and accepting state set $F \subseteq Q$. $\delta$ extends to trees: $\delta : T_\Sigma \longrightarrow Q$ is defined such that if $t = f[t_1, \ldots, t_n] \in T_\Sigma$ then $\delta(t) = \delta(\delta(t_1) \cdots \delta(t_n), f)$. The set of trees accepted by $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in T_\Sigma | \delta(t) \in F\}$ (a *regular* tree language).

It is well known that the Myhill-Nerode theorem carries over to regular tree languages: Let $L \subseteq T_\Sigma$. Given two trees $s, s' \in T_\Sigma$, let $s \sim_L s'$ iff for every $c \in C_\Sigma$, either both of $c[[s]]$ and $c[[s']]$ are in $L$ or none of them is. Obviously, $\sim_L$ is an equivalence relation on $T_\Sigma$. The equivalence class containing $s \in T_\Sigma$ is denoted by $[s]_L$. The *index* of $L$ equals $|\{[s]_L | s \in T_\Sigma\}|$. The Myhill-Nerode theorem states that $L$ is a regular tree language iff $L$ is of finite index iff $L$ is the union of all equivalence classes $[s]_L$ with $s \in L$. It follows from this that for every fta $\mathcal{A}$, $L(\mathcal{A})$ is of finite index. Conversely, if a tree language is of finite index, we can easily build an fta $\mathcal{A}_L$ recognizing $L$, with the states being the equivalence classes of $L$, $F = \{[s]_L | s \in L\}$, and, given some $f \in \Sigma_k$ and states $[s_1]_L, \dots, [s_k]_L$, $\delta_L([s_1]_L, \dots, [s_k]_L, f) = [f[s_1, \dots, s_k]]_L$. Moreover, this fta is the unique minimal fta recognizing $L$, up to a bijective renaming of states.

As indicated before, Drewes and Högberg [1] have designed a learning algorithm for regular tree languages, based on the one for strings by Angluin [2]. As strings can be seen as special trees over an alphabet containing only symbols of rank 1 or 0 ($\varepsilon$ being the only constant – the string *abc* is then noted as the expression $c(b(a(\varepsilon)))$, for example), this algorithm represents a generalization.

The aim of the learner is to construct an fta recognizing an unknown regular tree language $U \subseteq T_\Sigma$. He is helped by a teacher able to perform two tasks: The teacher can check whether $t \in U$ for some $t \in T_\Sigma$, and, given an fta $\mathcal{A}$, can return a $counterexample(\mathcal{A}) \in (U \setminus L(\mathcal{A})) \cup (L(\mathcal{A}) \setminus U)$. At any stage of the computation, the learner maintains two sets $S \subseteq T_\Sigma$ and $C \subseteq C_\Sigma$ satisfying certain conditions. Intuitively, one may imagine him building a table whose rows are indexed by the elements of $S \cup \Sigma(S)$ and the columns by the elements of $C$. The cell in row $s$ and column $c$ indicates whether $c[[s]] \in U$.

**Definition 1.** *The pair* $(S, C)$ *(*$S \subseteq T_\Sigma, C \subseteq C_\Sigma$ *finite,* $C$ *non-empty) is called an* observation table *if the following conditions hold:*

- *For every tree* $f[s_1, \dots, s_n]$*:* $s_1, \dots, s_n \in S$ *as well –* $S$ *is* subtree-closed, *and*
- *for every context* $c_0$ *of the form* $c[[f[s_1, \dots, s_{i-1}, \Box, s_{i+1}, \dots, s_n]]] \in C$*:* $c \in C$ *and* $s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n \in S$ *– we say that* $C$ *is* generalization-closed.

The elements of $S$ can be seen as candidates for representatives of the equivalence classes of $\sim_U$, and the elements of $C$ can be seen as witnesses that these representatives do indeed belong to different equivalence classes.

Given an observation table $T = (S, C)$ and a tree $s \in S \cup \Sigma(S)$, the observed behaviour of $s$ is denoted by $obs_T(s)$ (formally, $obs_T(s)$ denotes the function $obs : C \longrightarrow \{1, 0\}$ such that $obs(c) = 1$ iff $c[[s]] \in U$ for all $c \in C$).

**Definition 2.** *Observation table* $T = (S, C)$ *is* closed *if* $obs_T(\Sigma(S)) \subseteq obs_T(S)$*, and* consistent *if, for all* $f \in \Sigma_n$ *and all* $s_1, \dots, s_n, s'_1, \dots, s'_n \in S$*, if* $obs_T(s_i) = obs_T(s'_i)$ *for all* $i$ *with* $1 \leq i \leq n$ *then* $obs_T(f[s_1, \dots, s_n]) = obs_T(f[s'_1, \dots, s'_n])$*.*

These two properties are essential when building a candidate for the desired automaton: From a closed and consistent observation table $T = (S, C)$ one can synthesize an fta $\mathcal{A}_T$ whose set of states is $Q_T = \{obs_T(s) | s \in S\}$, the set of accepting states is $F_T = \{obs_T(s) | s \in S \cap U\}$, and $\delta_T(obs_T(s_1) \cdots obs_T(s_n), f) = obs_T(f[s_1, \dots, s_n])$ for all $f \in \Sigma_n$ and $s_1, \dots, s_n \in S$. Drewes and Högberg [1] formulate the following lemma, adapted from a corresponding one in [2]:

**Lemma 1.** *Let* $T = (S, C)$ *be a closed and consistent observation table. Then*

- $\delta(s) = obs_T(s)$ *for all* $s \in S \cup \Sigma(S)$*, and*
- *for all* $s \in S \cup \Sigma(S)$ *and all* $c \in C$*,* $\mathcal{A}_T$ *accepts* $c[[s]]$ *iff* $c[[s]] \in U$*.* $\mathcal{A}_T$ *is the unique minimal fta with this property (up to a bijective renaming of states).*

We prove a similar lemma for our generalized learning algorithm in Section 4.

The algorithm by Drewes and Högberg [1] can be seen below. $I$ is the index of $U$.[1] The learner starts with a table $T_1 = (\{a\}, \{\square\})$ (for some $a \in \Sigma_0$). The procedure CLOSURE adds suitable candidates to $S$ as long as $T$ is not closed (which corresponds to asking the teacher membership queries for these candidates and noting the result in the table). The procedure RESOLVE adds elements to $C$ as long as $T$ is not consistent. The procedure EXTEND synthesizes an fta from the current observation table (assume that an operation $synthesize(T)$ just exists) and asks the teacher for a counterexample. If the counterexample is unnecessarily large it is "pruned" via the procedure EXTRACT, which is an amelioration introduced by Drewes and Högberg [1] with respect to the original learner for strings by Angluin [2]. The counterexample is then added to the table, with its membership status.

```
$T = (S, C) := (\{a\}, \{\square\})$ for some arbitrary $a \in \Sigma_0$;
while $|\{obs_T(s) \mid s \in S\}| < I$ do
    if $T$ is not closed then $T$ := CLOSURE($T$)
    else if $T$ is not consistent then $T$ := RESOLVE($T$)
    else $T$ := EXTEND($T$)
end while;
return $\mathcal{A}_T$;

procedure CLOSURE($T$) where $T = (S, C)$
    find $s \in \Sigma(S)$ such that $obs_T(s) \notin obs_T(S)$;
    return $(S \cup \{s\}, C)$;

procedure RESOLVE($T$) where $T = (S, C)$
    find $c[[s]], c[[s']] \in \Sigma(S)$ where $s, s' \in S$ and $depth(c) = 1$ such that
        $obs_T(c[[s]]) \neq obs_T(c[[s']])$ and $obs_T(s) = obs_T(s')$;
    find $t, t' \in S$ such that
        $obs_T(t) = obs_T(c[[s]])$ and $obs_T(t') = obs_T(c[[s']])$;
    find $c' \in C$ such that $obs_T(t)(c') \neq obs_T(t')(c')$;
    return $(S, C \cup \{c'[[c]]\})$;

procedure EXTEND($T$) where $T = (S, C)$
    $\mathcal{A}_T$ := $synthesize(T)$;
    return EXTRACT($T, counterexample(\mathcal{A}_T)$);

procedure EXTRACT($T, t$) where $T = (S, C)$
    choose $c \in C_\Sigma$ and $s \in subtrees(t) \cap (\Sigma(S) \setminus S)$ such that $t = c[[s]]$;
    if there exists $s' \in S$ such that
        $obs_T(s') = obs_T(s)$ and $t \in U \Leftrightarrow c[[s']] \in U$ then
        return EXTRACT($T, c[[s']]$);
    else return $(S \cup \{s\}, C)$
    end if;
```

[1] In [1] $I$ is used as termination criterion. This does not affect the computation as such – they prove that the algorithm returns the desired automaton in time, i.e., it never halts without result because of that criterion alone – and is therefore equivalent to assuming that the teacher, when asked for a counterexample, first checks if $\mathcal{A} = \mathcal{A}_U$.

Let $T_l = (S_l, C_l)$ be the table obtained after $l-1$ steps. Note that according to Drewes and Högberg [1] (and as is easy to see) the procedures `CLOSURE`, `RESOLVE`, and `EXTEND` all guarantee that each constructed observation table satisfies the following conditions: (A) $T_l$ is indeed an observation table, (B) for all distinct trees $s, s' \in S_l$, $s \not\sim_U s'$, (C) $|S_l| + |C_l| = l + 1$, and (D) $|C_l| \leq |obs_{T_l}(S_l)|$.

Drewes and Högberg [1] prove that their algorithm always terminates after less than $2I$ loop executions and returns the desired fta (see [1] for the proof).

## 3 Multi-dimensional Trees and Automata

### 3.1 Multi-dimensional Trees as Defined by Rogers [3,4]

Starting from the tree definition based on two-dimensional tree domains, Rogers generalizes both downwards (to strings and points) and upwards and defines *labeled multi-dimensional trees* over a hierarchy of multi-dimensional tree domains:

**Definition 3 (Rogers [3,4]).** *Let ${}^d1$ be the class of all dth-order sequences of* $1$*s:* ${}^01 := \{1\}$*, and* ${}^{n+1}1$ *is the smallest set satisfying (i)* $\langle\rangle \in {}^{n+1}1$*, and (ii) if* $\langle x_1, \ldots, x_l\rangle \in {}^{n+1}1$ *and* $y \in {}^n1$*, then* $\langle x_1, \ldots, x_l, y\rangle \in {}^{n+1}1$*. Let* $\mathbb{T}^0 := \{\emptyset, \{1\}\}$ *(point domains). A* $(d+1)$*-dimensional tree domain is a set of hereditarily prefix closed* $(d+1)$*st-order sequences of* $1$*s, i.e.,* $\mathsf{T} \in \mathbb{T}^{d+1}$ *iff*

- $\mathsf{T} \subseteq {}^{d+1}1$,
- $\forall s, t \in {}^{d+1}1 : s \cdot t \in \mathsf{T} \Rightarrow s \in \mathsf{T}$,
- $\forall s \in {}^{d+1}1 : \{w \in {}^d1 | s \cdot \langle w\rangle \in \mathsf{T}\} \in \mathbb{T}^d$.

*A* $\Sigma$*-labeled* $\mathsf{T}d$ *(*$d$-dimensional tree*) is a pair* $\langle T, \tau\rangle)$ *where* $T$ *is a* $d$*-dimensional tree domain and* $\tau : T \longrightarrow \Sigma$ *is an assignment of labels in the (non-partitioned) alphabet* $\Sigma$ *to nodes in* $T$*. We will denote the class of all* $\Sigma$*-labeled* $\mathsf{T}d$ *as* $\mathbb{T}^d_\Sigma$*.*

Every $d$-dimensional tree can be conceived to be built up from one or more $d$-dimensional *local trees*, that is, trees of depth at most one in their major dimension. Each of these smaller trees consists of a root and an arbitrarily large $(d-1)$-dimensional "child tree" consisting of the root's children (a formal definition of the set $\mathbb{T}^{d,loc}_\Sigma$ of all local trees over some alphabet $\Sigma$ would be for example $\mathbb{T}^{d,loc}_\Sigma = \{\langle T, \tau\rangle | \langle T, \tau\rangle$ is a $\Sigma$-labeled $\mathsf{T}d$, and $\forall s \in T : |s| \leq 1\})$. Local strings (i.e., one-dimensional trees), for example, consist of a root and a point as its child tree. Local two-dimensional trees consist of a root and a string. Local three-dimensional trees would have a pyramidal form, with a two-dimensional tree as its base. There are also trivial local trees (consisting of a single root), and even empty ones. Composite trees can be built from local ones by identifying the root of one local tree with a node in the child tree of another (and adapting the addresses in order to incorporate them into the newly created tree domain). Figure 1 shows examples of local and composite trees for the first four steps of the hierarchy (only some composite trees are labeled, and in the three-dimensional case, only the addresses of nodes that do not appear in the rightmost local tree as well are given, for clarity. $\varepsilon_d$ denotes an empty sequence of order $d$).

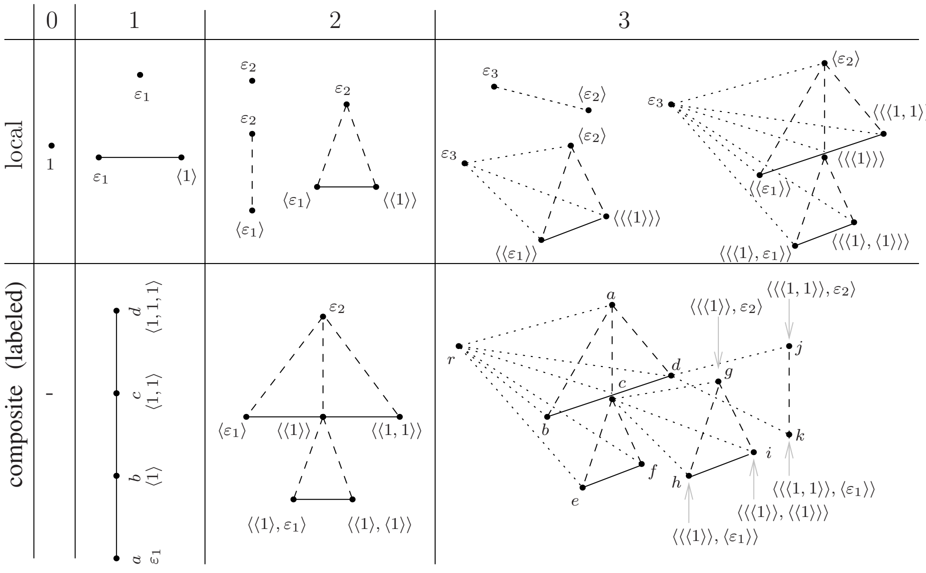

0
1
2
3
local
composite (labeled)
1
-
$\varepsilon_1$
$\langle 1\rangle$
$\varepsilon_2$
$\langle\varepsilon_1\rangle$
$\langle\langle 1\rangle\rangle$
$\varepsilon_3$
$\langle\varepsilon_2\rangle$
$\langle\langle\varepsilon_1\rangle\rangle$
$\langle\langle\langle 1\rangle\rangle\rangle$
$\langle\langle\langle 1,1\rangle\rangle\rangle$
$\langle\langle\langle 1\rangle,\varepsilon_1\rangle\rangle$
$\langle\langle\langle 1\rangle,\langle 1\rangle\rangle\rangle$
$a$ $\varepsilon_1$
$b$ $\langle 1\rangle$
$c$ $\langle 1,1\rangle$
$d$ $\langle 1,1,1\rangle$
$\langle\langle 1,1\rangle\rangle$
$\langle\langle 1\rangle,\varepsilon_1\rangle$
$\langle\langle 1\rangle,\langle 1\rangle\rangle$
$r$ $a$ $b$ $c$ $d$ $e$ $f$ $g$ $h$ $i$ $j$ $k$
$\langle\langle\langle 1\rangle\rangle,\varepsilon_2\rangle$
$\langle\langle\langle 1,1\rangle\rangle,\varepsilon_2\rangle$
$\langle\langle\langle 1,1\rangle\rangle,\langle\varepsilon_1\rangle\rangle$
$\langle\langle\langle 1\rangle\rangle,\langle\langle 1\rangle\rangle\rangle$
$\langle\langle\langle 1\rangle\rangle,\langle\varepsilon_1\rangle\rangle$

**Fig. 1.** Local and composite trees for $d = 0, 1, 2, 3$

**Definition 4 (Rogers [4]).** *A* $\mathsf{T}d$ *automaton with finite state set* $Q$ *and (non-ranked) alphabet* $\Sigma$ *is a finite set of triples* $\mathcal{A}^d \subseteq \Sigma \times Q \times \mathbb{T}_Q^{d-1}$.

The interpretation of a triple $\langle\sigma, q, \mathcal{T}\rangle \in \mathcal{A}^d$ is that if a node of a $\mathsf{T}d$ is labeled with $\sigma$ and $\mathcal{T}$ encodes the assignment of states to its children, that node may be assigned state $q$. A *run* of a $\mathsf{T}d$ automaton on a $\Sigma$-labeled $\mathsf{T}d$ $\mathcal{T} = \langle T, \tau\rangle$ is a mapping $r : T \longrightarrow Q$ in which each assignment is licensed by $\mathcal{A}^d$. Note that a maximal node (wrt the major dimension, i.e., a leaf) labeled with $\sigma$ may be assigned state $q$ only if there is a triple $\langle\sigma, q, \emptyset\rangle \in \mathcal{A}^d$. If $F \in Q$ is the set of accepting states, then the set of (finite) $\Sigma$-labeled $\mathsf{T}d$ recognized by $\mathcal{A}^d$ is that set for which there is a run of $\mathcal{A}^d$ that assigns the root a state in $F$.

$\mathsf{T}1$ automata correspond to finite-state automata for strings, they recognize the regular languages. $\mathsf{T}2$ automata correspond to (non-deterministic) finite-state automata for trees, i.e., they recognize the regular tree languages, the associated string sets of which are the context-free languages. For $d \geq 3$, $\mathsf{T}d$ automata recognize languages of $d$-dimensional trees whose sets of string yields are situated between the classes of context-free and context-sensitive languages in the Chomsky Hierarchy, where for every $d$ the class of string yields of the $d$-dimensional tree languages is properly contained in the next (i.e., for $d + 1$).

### 3.2 Multi-dimensional Trees as Terms

In this subsection we will give a representation for multi-dimensional trees (first defined in [9]) which establishes them as a direct generalization of the one on which (classical) finite-state tree automata are based, i.e., one that allows multi-dimensional trees to be noted as expressions over a partitioned alphabet.

We use finite $d$-dimensional tree labeling alphabets $\Sigma^d$ where each symbol $f \in \Sigma^d$ is associated with at least one unlabeled $(d-1)$-dimensional tree $t$ specifying the admissible child structure for a root labeled with $f$ (note that as before it is just as possible to associate several, albeit finitely many, admissible child structure trees with one symbol). $t$ can be given in any form suitable for trees, as long as it is compatible with the existence of an empty tree. For consistency's sake we will use the definition of multi-dimensional trees given below and write $t$ as an expression over a special kind of "alphabet" containing just one symbol $\rho$ for which any child structure is admissible.

Let $\Sigma_t^d$ for $d \geq 1$ be the set of all symbols associated with $t$ and $\Sigma^0$ a set of constant symbols. The set of all $d$-dimensional trees $\mathbb{T}_{\Sigma^d}$ can then be defined:

**Definition 5.** *Let $\varepsilon^d$ be the empty $d$-dimensional tree. Then*

- $\mathbb{T}_{\Sigma^0} := \{\varepsilon^0\} \cup \Sigma^0$, *and*
- *for $d \geq 1$: $\mathbb{T}_{\Sigma^d}$ is the smallest set such that $\varepsilon^d \in \mathbb{T}_{\Sigma^d}$ and $f[t_1, \dots, t_n]_t \in \mathbb{T}_{\Sigma^d}$ for every $f \in \Sigma_t^d$, $n$ the number of nodes in $t$, $t_1, \dots, t_n \in \mathbb{T}_{\Sigma^d}$ and $t_1, \dots, t_n$ are rooted breadth-first in that order*[2] *at the nodes of $t$.*

Multi-dimensional trees in this notation can be translated one-to-one into trees in Rogers' notation and vice versa – see [9] for the translation and proof.

For $t_p = f[t_1, \dots, t_n]_t$ with $f \in \Sigma_t^d$, $t_1, \dots, t_n$ are the direct subtrees of $t_p$, $subtrees(t)$ is defined as in Section 2. Also, for some fixed $d$, let $\Box$ be a special symbol associated with $\varepsilon^{d-1}$ (i.e., a leaf label). A tree $c \in \mathbb{T}_{\Sigma^d \cup \{\Box\}}$ in which $\Box$ occurs exactly once is still called a context, the set of contexts over $\Sigma^d$ is $\mathbb{C}_{\Sigma^d}$. $c[[s]]$ for $c \in \mathbb{C}_{\Sigma^d}$ and $s \in \mathbb{T}_{\Sigma^d}$ is defined via substitution as before.

We can represent $d$-dimensional automata as a direct generalization of fta's:

**Definition 6.** *A (total, deterministic) finite-state $d$-dimensional tree automaton is a quadruple $\mathcal{A}^d = (\Sigma^d, Q, \delta, F)$ with input alphabet $\Sigma^d$, finite state set $Q$, set of accepting states $F \subseteq Q$ and transition function $\delta$ with with $\delta(t(q_1, \dots, q_n), f) \in Q$ for every $f \in \Sigma_t^d$ where $t(q_1, \dots, q_n)$ encodes the assignment of states to the nodes of $t$ (i.e., $t(q_1, \dots, q_n)$ is isomorphic to $t$ and its nodes are labeled with $q_1, \dots, q_n$ breadth-first in that order if $Q$ is taken as a partitioned alphabet in which every element is associated with all the child structures it occurs with in $\delta$). The transition function extends to $d$-dimensional trees: $\delta : \mathbb{T}_{\Sigma^d} \longrightarrow Q$ is defined such that if $t_p = f[t_1, \dots, t_n]_t \in \mathbb{T}_{\Sigma^d}$ then $\delta(t_p) = \delta(t(\delta(t_1), \dots, \delta(t_n)), f)$. The set of trees accepted by $\mathcal{A}^d$ is $L(\mathcal{A}^d) = \{t_p \in \mathbb{T}_{\Sigma^d} | \delta(t_p) \in F\}$.*

The equivalence between this definition and the one by Rogers [4] is easy to see. For two corresponding automata $\mathcal{A}^d = (\Sigma^d, Q, \delta, F)$ and $\mathcal{A}_R^d \subseteq \Sigma_R \times Q_R \times \mathbb{T}_{Q_R}^{d-1}$ (accepting state set $F_R$) in the two notations the sets of states $Q$ and $Q_R$, and $F$ and $F_R$ coincide, the construction of $\Sigma_R$ from $\Sigma^d$ is trivial, and $\Sigma^d$ is constructed from $\mathcal{A}_R^d$ as follows: $f \in \Sigma_t^d$ for all triples $\langle f, q, t_0 \rangle \in \mathcal{A}_R^d$,

[2] This is an ad hoc settlement, any other spatial arrangement would do as well.

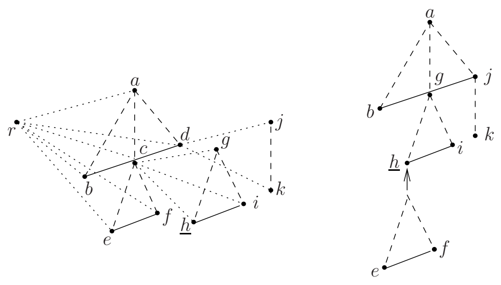


**Fig. 2.** Ambiguity in the yield for $d \geq 3$ (resolved by marked splicing points)

where $t \in \mathbb{T}_{\{\rho\}^{d-1}}$ is isomorphic to $t_0$. Most importantly, there is a one-to-one correspondence between the elements of $\mathcal{A}_R^d$ and $\delta$: Every triple $\langle f, q, t_0 \rangle \in \mathcal{A}_R^d$ can be translated to an assignment $\delta(\Psi(t_0), f) = q$ of $\mathcal{A}^d$, and every assignment $\delta(t(q_1, \ldots, q_n), f) = q$ of $\mathcal{A}^d$ to a triple $\langle f, q, \Phi(t(q_1, \ldots, q_n)) \rangle \in \mathcal{A}_R^d$, where $\Phi$ and $\Psi$ are translations from one notation into the other (see [9] for a definition). From this and from the identical state sets it follows that $L(\mathcal{A}_R^d) = \Psi(L(\mathcal{A}^d))$ and $L(\mathcal{A}^d) = \Phi(L(\mathcal{A}_R^d))$.

Finally, we give a definition of the yield operation for multi-dimensional trees in the new notation. As for $d \geq 3$ the yield is not unambiguous (see Figure 2), the structures have to be enriched with additional information. Assume that, for $d \geq 2$, in every tree $t_p \in \mathbb{T}_{\Sigma^d}$ every labeling symbol $f \in \Sigma^d$ is indexed with a set $S \subseteq \{2, \ldots, d\}$. If $x \in S$ then we call a node labeled by $f_S$ a *foot node for the* $(x-1)$*-dimensional yield of* $t_p$. For every subtree $t_q$ of $t_p$ the distribution of these foot nodes must fulfil certain conditions:

(1) If $t_q$ has depth 0 the index set in its root label must contain $d$, otherwise $t_q = f_S[t_1, \ldots, t_n]_t$ with $f \in \Sigma_t^d$, $S \subseteq \{2, \ldots, d\}$, and $t_1, \ldots, t_n \in \mathbb{T}_{\Sigma^d}$ must have exactly one direct subtree $t_i \in \{t_1, \ldots, t_n\}$ whose root labeling symbol is indexed with a set containing $d$ and this subtree is attached to a *leaf* in $t$. In both cases, we will refer to that root as the $d$-dimensional foot node of $t_q$.
(2) The foot nodes are distributed in such a way that for every $n$-dimensional yield of $t_p$ with $n < d$, condition (1) is fulfilled as well.

For $d \geq 2$, the $(d-1)$-dimensional yield of a tree $t_p \in \mathbb{T}_{\Sigma^d}$ is defined as

$$yd_{d-1}(t_p) = \begin{cases} \varepsilon^{d-1} & \text{for } t_p = \varepsilon^d, \\ a_S & \text{for } t_p = a_S \text{ with } a \in \Sigma_{\varepsilon^{d-1}}^d \text{ and } S \subseteq \{2, \ldots, d\}, \\ op_{t_p}(t_1) & \text{for } t_p = f_S[t_1, \ldots, t_n]_t \text{ with } t_1, \ldots, t_n \in \mathbb{T}_{\Sigma^d},\ f \in \Sigma_t^d, \\ & t \neq \varepsilon^{d-1}, \text{ and } S \subseteq \{2, \ldots, d\}, \end{cases}$$

where $op_{t_p}(t_i)$ for $t_i \in \{t_1, \ldots, t_n\}$ is defined as the $(d-1)$-dimensional tree that is obtained by replacing the $d$-dimensional foot node of $t_i$ in $yd_{d-1}(t_i)$ by $e_R[op_{t_p}(t_j), \ldots, op_{t_p}(t_k)]_{t_x}$, where $e_R$ with $e \in \Sigma^d$ and $R \subseteq \{2, \ldots, d\}$ is the label of the foot node, $t_x$ is the $(d-2)$-dimensional child structure of the node at which $t_i$ is attached in $t$ and $t_j, \ldots, t_k$ are the direct subtrees of $t_p$ that are attached (breadth-first in that order) at the nodes of $t_x$. The result $yd_{d-1}(t_p)$ is a $(d-1)$-dimensional tree over an alphabet $\Sigma^{d-1}$ containing at least all the node labels in $yd_{d-1}(t_p)$, each associated at least with the child structures it occurs with. Obviously, the string yield of a $d$-dimensional tree for $d \geq 2$ can be obtained by taking the $(d-1)$-dimensional yield $d-1$ times.

Example 1 defines an automaton $\mathcal{A}^3_{ww}$ recognizing a three-dimensional tree language whose set of string yields $yd_1(L(\mathcal{A}^3_{ww}))$ is $L_{ww} = \{ww | w \in \{a,b\}^+\}$:

*Example 1.* $\mathcal{A}^3_{ww} = (\Sigma^3, \{q_a, q_b, q_g, q_y, q_z, q_f, q_x\}, \delta, \{q_f\})$ where $\Sigma^3 = \{a, b, f, g, h_{\{3\}}\}$ with $a, b, f, g, h_{\{3\}} \in \Sigma^3_{\varepsilon^2}$ and $f \in \Sigma^3_{t_1}$ for $t_1 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^0}]_{\rho[]_{\varepsilon^0}}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$ and $f \in \Sigma^3_{t_2}$ for $t_2 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^1}, \rho[]_{\varepsilon^1}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$. (Note that in $\Sigma^3$ only index sets containing 3 have been given, as the distribution of foot nodes for the string yield is never ambiguous). $\delta$ is defined as follows:

$$\begin{aligned} \delta(\varepsilon^2, a) &= q_a & \delta(t_1(q_g, q_a, q_z, q_a)) &= q_f \\ \delta(\varepsilon^2, b) &= q_b & \delta(t_1(q_g, q_b, q_z, q_b)) &= q_f \\ \delta(\varepsilon^2, f) &= q_z & \delta(t_2(q_g, q_a, q_z, q_y, q_a)) &= q_z \\ \delta(\varepsilon^2, g) &= q_g & \delta(t_2(q_g, q_b, q_z, q_y, q_b)) &= q_z \\ \delta(\varepsilon^2, h_{\{3\}}) &= q_y \end{aligned}$$

and $\delta(t_0, x) = q_x$ for all other admissible $t_0$ and all symbols $x \in \Sigma^3$. Figure 3 shows $t_1$ and $t_2$, three trees $t_a, t_b, t_c \in L(\mathcal{A}^3_{ww})$ in the middle, and the two-dimensional yield for $t_c$, whose one-dimensional yield is the string *abab*.

With the slightly adapted definitions of contexts and automata, the Myhill-Nerode theorem (see Section 2) carries over quite naturally to multi-dimensional trees, and consequently, for every recognizable $d$-dimensional tree language $L$ there exists a unique minimal automaton $\mathcal{A}^d_L$ recognizing $L$. It is this fact that enables us to give a learning algorithm for languages of trees of arbitrarily many dimensions based on the same principle as the one by Drewes and Högberg [1].

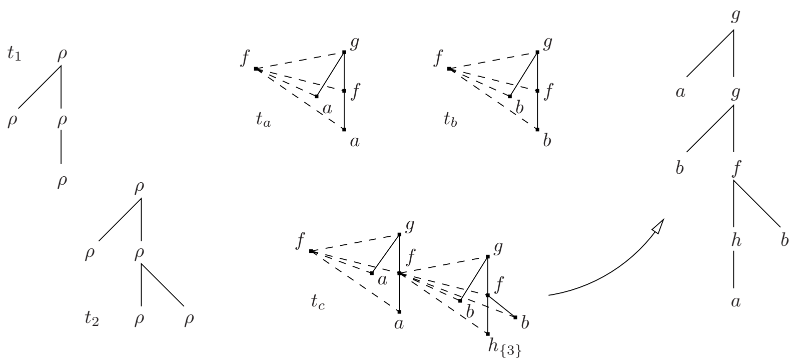


**Fig. 3.** Example 1

## 4 The Learner for Multi-dimensional Tree Languages

We will now generalize the learning algorithm for regular tree languages by Drewes and Högberg [1] to recognizable tree languages of arbitrarily many dimensions. The necessary concepts for the learning algorithm can be adapted in an obvious way (assume that $t$ has $n$ nodes):

- *Subtree-closed*: For every tree $f[t_1, \dots, t_n]_t \in S$: $t_1, \dots, t_n \in S$.
- *Generalization-closed*: For every context of the form $c[[f[t_1, \dots, t_{i-1}, \Box, t_{i+1}, \dots, t_n]_t]]$, $c$ is in $C$ as well and $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$ are in $S$.
- $\Sigma^d(S)$ denotes the set of all trees of the form $f[t_1, \dots, t_n]_t$ such that $f \in \Sigma^d_t$ for some $t$ and $t_1, \dots, t_n \in S$.
- *Closed (for an observation table $T$)*: $T$ is closed if $obs_T(\Sigma^d(S)) \subseteq obs_T(S)$.
- *Consistent (for an observation table $T$)*: For all $f \in \Sigma^d_t$ and all $s_1, \dots, s_n$, $s'_1, \dots, s'_n \in S$, if $obs_T(s_i) = obs_T(s'_i)$ for all $i$ with $1 \le i \le n$ then $obs_T(f[s_1, \dots, s_n]_t) = obs_T(f[s'_1, \dots, s'_n]_t)$.

From a closed and consistent observation table $T = (S, C)$ one can derive a $d$-dimensional tree automaton $\mathcal{A}^d_T$ whose set of states is $Q_T = \{obs_T(s) | s \in S\}$, the set of accepting states is $F_T = \{obs_T(s) | s \in S \cap U\}$, and $\delta_T(t(obs_T(s_1), \dots, obs_T(s_n)), f) = obs_T(f[s_1, \dots, s_n]_t)$ for all $f \in \Sigma^d_t$ and $s_1, \dots, s_n \in S$.

With this settled, the learning algorithm for recognizable $d$-dimensional tree languages (not containing the empty tree, and for some $d \ge 1$, since $\mathbb{T}_{\Sigma^0}$ is finite, i.e., trivial to learn) is very easy to formulate; in fact, it is identical to the one given in Section 2 (just change $\Sigma(S)$ to $\Sigma^d(S)$ throughout and start with $T = (\{a\}, \{\Box\})$ for some arbitrary $a \in \Sigma^d_{\varepsilon^{d-1}}$). We will prove that the validity of Lemma 1, repeated below in a slightly adapted form as Lemma 2, hasn't been changed by our generalization to trees of arbitrarily many dimensions. The proofs are inspired by the corresponding ones in [2].

**Lemma 2.** *Let $T = (S, C)$ be a closed, consistent observation table. Then*

*(a) $\delta_T(t_p) = obs_T(t_p)$ for all $t_p \in S \cup \Sigma^d(S)$,*
*(b) for all $t_p \in S \cup \Sigma^d(S)$ and all contexts $c \in C$, $\mathcal{A}^d_T$ accepts $c[[t_p]]$ iff $c[[t_p]] \in U$,*
*(c) $\mathcal{A}^d_T$ is the unique minimal automaton with property (b).*

*Proofs.* (a) is proved by induction via the definitions of $\delta$ and $\delta_T$: It certainly holds for all $a \in \Sigma^d_{\varepsilon^{d-1}} \cap S$ (trees consisting of one node in $S$), as $\delta_T(a) = \delta_T(\varepsilon^{d-1}, a) = obs_T(a)$. Now let $t_p = f[s_1, \dots, s_n]_t$ for an arbitrary $f \in \Sigma^d_t$ and $s_1, \dots, s_n \in S \cup \Sigma^d(S)$. As $t_p \in S \cup \Sigma^d(S)$, $s_1, \dots, s_n$ must be in $S$ (which is clear for $t_p \in \Sigma^d(S)$ – for $t_p \in S$ recall that $S$ is subtree-closed). If (a) holds for $s_1, \dots, s_n$ then it also holds for $t_p$, as $\delta_T(f[s_1, \dots, s_n]_t) = \delta_T(t(\delta_T(s_1), \dots, \delta_T(s_n)), f) = \delta_T(t(obs_T(s_1), \dots, obs_T(s_n)), f) = obs_T(f[s_1, \dots, s_n]_t)$. ■

(b) is proved by induction over the depth of the contexts in $C$. For $c = \Box$ and all $t_p \in S \cup \Sigma^d(S)$, $\delta_T(c[[t_p]]) = \delta_T(t_p) = obs_T(t_p)$ by (a). $t_p$ is either in $S$ or in $\Sigma^d(S)$. Case 1, $t_p \in S$: $\delta_T(t_p) \in F \Leftrightarrow obs_T(t_p) \in F \Leftrightarrow obs_T(t_p) \in \{obs_T(s) | s \in S \cap U\} \Leftrightarrow t_p \in S \cap U \Leftrightarrow t_p \in U$. Case 2, $t_p \in \Sigma^d(S)$: As $T$ is closed, there exists

$t_q \in S$ with $obs_T(t_p) = obs_T(t_q)$, and thus $\delta_T(t_p) = \delta_T(t_q)$ (and the rest of the argument runs as in Case 1). This proves (b) for $c = \Box$.

Assume that $c \in C$ is of depth $k+1$, and (b) holds for all contexts in $C$ up to depth $k$ and all $t_p \in S \cup \Sigma^d(S)$. As $C$ is generalization-closed, there exists $c_2 \in C$ of depth $k$ and $s_1, \ldots, s_n \in S$ such that $c = c_2[[f[s_1, \ldots, s_{i-1}, \Box, s_{i+1}, \ldots, s_n]_t]]$ for some $f \in \Sigma_t^d$. (b) holds for $c_2$: $\delta_T(c_2[[t_p]]) \in F \Leftrightarrow c_2[[t_p]] \in U$ for all $t_p \in S \cup \Sigma^d(S)$. As $T$ is closed, there exists $t_q \in S$ with $\delta_T(t_p) = obs_T(t_p) = obs_T(t_q) = \delta_T(t_q)$. Obviously, $f[s_1, \ldots, s_{i-1}, t_q, s_{i+1}, \ \ldots, s_n]_t$ is in $\Sigma^d(S)$. With $\delta_T(t_q) = \delta_T(t_p)$ and $T$ consistent, $\delta_T(c_2[[f[s_1, .., s_{i-1}, t_q, s_{i+1}, .., s_n]_t]]) = \delta_T(c_2[[f[s_1, .., s_{i-1}, t_p, s_{i+1}, .., \ s_n]_t]]) \in F \Leftrightarrow c_2[[f[s_1, \ldots, s_{i-1}, t_p, s_{i+1}, \ldots, \ s_n]_t]] \in U \Leftrightarrow c[[t_p]] \in U$, which proves (b) for all $c \in C$ and all $t_p \in S \cup \Sigma^d(S)$. ■

(c) is equivalent to the claim that any automaton $\mathcal{A}^{d'} = (\Sigma^d, Q', \delta', F')$ consistent with $T$ with as many or fewer states than $\mathcal{A}_T^d$ is isomorphic to $\mathcal{A}_T^d$. Let $\mathcal{A}_T^d$ have $n$ states. Define, for each $q' \in Q'$, $obs_T(q')$ as the finite function $g : C \longrightarrow \{0, 1\}$ such that $g(c) = 1$ iff $\delta'(c[[q']]) \in F'$, where $\delta'(c[[q']]) = \delta'(c[[t]])$ for all $t$ with $\delta'(t) = q'$. Since $\mathcal{A}^{d'}$ is consistent with $T$, for each $t_p \in S \cup \Sigma^d(S)$ and each $c \in C$, $\delta'(c[[t_p]]) \in F'$ iff $obs_T(t_p)(c) = 1$, which also means that $\delta'(c[[\delta'(t_p)]]) \in F'$ iff $obs_T(t_p)(c) = 1$, so $obs_T(\delta'(t_p)) = obs_T(t_p)$. As $t_p$ ranges over all of $S$, $obs_T(\delta'(t_p))$ ranges over all of $Q$, so $\mathcal{A}^{d'}$ must have $n$ states.

Thus, for each $t_p \in S$ there is a unique $q' \in Q'$ such that $obs_T(t_p) = obs_T(q')$, namely $\delta'(t_p)$. Define for each $t_p \in S$, $\phi(obs_T(t_p)) = \delta'(t_p)$. This mapping is bijective. We must verify that it preserves $\delta$ and maps $F$ to $F'$. For $s_1, \ldots, s_n \in S$ and $f \in \Sigma_t^d$, let $t_p \in S$ such that $obs_T(f[s_1, \ldots, s_n]_t) = obs_T(t_p)$. Then $\phi(\delta(t(obs_T(s_1), \ldots, obs_T(s_n)), f)) = \phi(obs_T(f[s_1, \ldots, s_n]_t)) = \phi(obs_T(t_p)) = \delta'(t_p)$, and $\delta'(t(\phi(obs_T(s_1)), \ldots, \phi(obs_T(s_n))), f) = \delta'(t(\delta'(s_1), \ldots, \delta'(s_n)), f) = \delta'(f[s_1, \ldots, s_n]_t)$. Since $obs_T(\delta'(t_p)) = obs_T(\delta'(f[s_1, \ldots, s_n]_t))$, $\delta'(t_p)$ and $\delta'(f[s_1, \ldots, s_n]_t)$ must be the same state of $\mathcal{A}^{d'}$, we can conclude that $\phi(\delta(t[obs_T(s_1), \ldots, obs_T(s_n)], f)) = \delta'(t[\phi(obs_T(s_1)), \ldots, \phi(obs_T(s_n))], f)$ for all $s_1, \ldots, s_n \in S$ and $f \in \Sigma_t^d$. To complete the proof we must see that $\phi$ maps $F$ to $F'$: This is clear since if $obs_T(t_p) \in F$ then $t_p \in U$ for all $t_p \in S$, so as $\phi(obs_T(t_p))$ is mapped to a state $q'$ with $obs_T(q') = obs_T(t_p)$, $q'$ must be in $F$. Conversely, if $obs_T(t_p)$ is mapped to a state $q' \in F'$, then since $obs_T(q') = obs_T(t_p)$, $t_p \in U$, so $obs_T(t_p) \in F$. $\phi$ is indeed an isomorphism, and (c) is proved. ■

**Theorem 1.** *The learner returns $\mathcal{A}_U^d$ after less than $2I$ loop executions.*

The proof stays as in [1]: According to property (D) of the obtained observation tables there cannot be more contexts in $C_l$ than trees in $S_l$, for all $l$. Since (C) states that $|C_l| + |S_l| = l + 1$, this means $|S_l| > l/2$. We also know that the learner halts when $S_l$ has $I$ elements (by (B)), so it will halt before $l = 2I$.

Let $\mathcal{A}_{T_m}^d$ be the returned automaton. $T_m$ is a closed, consistent observation table and $\mathcal{A}_{T_m}^d$ is the unique minimal automaton such that, for all $s \in S_m$ and $c \in C_m$, $c[[s]] \in L(\mathcal{A}_{T_m}^d)$ iff $c[[s]] \in U$ (Lemma 2(b)). However, $\mathcal{A}_U^d$ has the same property and as many states, so $\mathcal{A}_{T_m}^d = \mathcal{A}_U^d$ up to a bijective state renaming. ■

We sketch a run of the algorithm for the language $L(\mathcal{A}_{ww}^3)$ from Example 1.

*Example 2.* The learner starts with $T_1 = (\{a\}, \{\Box\})$, and $obs_T(a) = 0$. $T_1$ is closed and consistent, so the learner proposes $\mathcal{A}_{T_1}$, which accepts nothing, and gets the counterexample $t_b = f[g, b, f, b]_{t_1}$ (see Figure 3), from which the procedure `EXTRACT` derives $T_2$. $T_2$ is closed, but not consistent (for example, $obs_T(a) = 0$ and $obs_T(b) = 0$, but $obs_T(c[[a]]) = 0$ and $obs_T(c[[b]]) = 1$ for $c = f[g, b, f, \Box]_{t_1}$). Several invocations of `RESOLVE` yield $T_3$, from which the learner synthesizes an automaton $\mathcal{A}_{T_3}$ with five states and one accepting state $obs_T(t_b)$, and gets the counterexample $t_c = f[g, a, f[g, b, f, h_{\{3\}}, b]_{t_2}, a]_{t_1}$ (see Figure 3). The procedure `EXTRACT` adds two more rows to $T_3$, and `RESOLVE` another column, yielding $T_4$, which is closed and consistent. $\mathcal{A}_{T_4}$ has $I = 7$ states (which is the termination criterion, see Section 2) and recognizes $L(\mathcal{A}^3_{ww})$.

| $T_2$ | $\Box$ |
|---|---|
| $a$ | 0 |
| $b$ | 0 |
| $f$ | 0 |
| $g$ | 0 |
| $t_b$ | 1 |

| $T_3$ | $\Box$ | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|---|
| $a$ | 0 | 0 | 0 | 0 |
| $b$ | 0 | 1 | 0 | 0 |
| $f$ | 0 | 0 | 1 | 0 |
| $g$ | 0 | 0 | 0 | 1 |
| $t_b$ | 1 | 0 | 0 | 0 |

| $T_4$ | $\Box$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ |
|---|---|---|---|---|---|
| $a$ | 0 | 0 | 0 | 0 | 0 |
| $b$ | 0 | 1 | 0 | 0 | 0 |
| $f$ | 0 | 0 | 1 | 0 | 0 |
| $g$ | 0 | 0 | 0 | 1 | 0 |
| $t_b$ | 1 | 0 | 0 | 0 | 0 |
| $h_{\{3\}}$ | 0 | 0 | 0 | 0 | 1 |
| $t_c$ | 1 | 0 | 1 | 0 | 0 |

$t_b = f[g, b, f, b]_{t_1}$
$t_c = f[g, a, f[g, b, f, h_{\{3\}}, b]_{t_2}, a]_{t_1}$
$t_1 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^0}]_{\rho[]_{\varepsilon^0}}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$
$t_2 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^1}, \rho[]_{\varepsilon^1}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$
$c_1 = f[g, b, f, \Box]_{t_1}$
$c_2 = f[g, b, \Box, b]_{t_1}$
$c_3 = f[\Box, b, f, b]_{t_1}$
$c_4 = f[g, a, f[g, b, f, \Box, b]_{t_2}, a]_{t_1}$

We have shown that the algorithm by Drewes and Högberg [1] can be used in an almost unchanged form to learn multi-dimensional trees in the new notation introduced in Subsection 3.2. This is tantamount to the claim that the algorithm is also able to learn even string languages that lie beyond the context-free class, provided that the learned multi-dimensional tree languages are enriched with the information that is needed in order to take the yields. Probably the easiest way to do this is to integrate the index sets directly into the alphabet (as has been done in Example 1), i.e., to multiply the symbols of the alphabet by the power set of $S^d = \{2, ..., n\}$ for $d \geq 2$. String languages situated beyond the regular class can then be learned in a two-step approach by first letting the algorithm learn a higher-dimensional representation of the language and then taking the string yields of the set that is recognized by the resulting automaton.

## 5 Conclusion

Generalizing the MAT learning algorithm $L^*$ to regular tree languages of arbitrarily many dimensions is only a first step to a more thorough understanding of the interaction between grammatical inference and formal language theory. The next steps would be to find $L^*$-like learning algorithms for finite-state recognizable languages of all kinds of objects, such as for example graphs or pictures, or take existing ones, such as the learning algorithm $L^\omega$ for $\omega$-regular string languages [10], and try to integrate these often very similar looking algorithms into a single one that can process as many different types of inputs as possible. The same can then be attempted for other learning models and algorithms.

Ultimately, it is our goal to understand which general mathematical properties of formal language classes of all kinds of suitable objects underlie algorithmical learnability (under a certain model). Are they best captured in terms of universal algebra, or mathematical logics? At least for the class of regular languages, which up until now is the most explored formal language class in the area of grammatical inference, there is some evidence pointing to universal algebra as a convenient foundation, such as the Myhill-Nerode theorem or the fact that finite-state automata can be best defined for objects with term-like representations. Since we do not want to restrict ourselves to string or tree languages, this opens up another interesting question: What are the exact properties (if they can be formulated at all) that characterize the term of "regularity" in general?

To come back to the results of our paper: Possibly the finding that recognizable three-dimensional tree languages are learnable and the fact that these are connected to the linguistically inspired grammar formalism TAG can bring about more research and consequently more knowledge about natural language learning as well. The connection between formal learnability and human language acquisition is an ample field of speculations which are yet to be verified.

Another goal for the near future would be to try and implement the results of this paper. As an implementation for the algorithm by Drewes and Högberg [1] already exists, this should not be too hard to accomplish. Such a project might also be an impulse to reflect further on complexity issues (see also [1,11]).

## References

1. Drewes, F., Högberg, J.: Learning a Regular Tree Language from a Teacher. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 279–291. Springer, Heidelberg (2003)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
3. Rogers, J.: Syntactic Structures as Multi-dimensional Trees. Research on Language and Computation 1, 265–305 (2003)
4. Rogers, J.: wMSO Theories as Grammar Formalisms. TCS 293, 291–320 (2003)
5. Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theoretical Computer Science 76(2–3), 223–242 (1990)
6. Joshi, A.K.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description. In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Processing. Cambridge University Press, Cambridge (1985)
7. Weir, D.J.: A geometric hierarchy beyond context-free languages. Theoretical Computer Science 104(2), 235–261 (1992)
8. Kasprzik, A.: Two Equivalent Regularizations of Tree Adjoining Grammars. Technical Report 08-1, University of Trier (2008), `urts117.uni-trier.de/cms/index.php?id=15939`
9. Kasprzik, A.: Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-dimensional Trees. Technical Report 08-3, University of Trier (2008), `urts117.uni-trier.de/cms/index.php?id=15939`
10. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. In: Proc. 4th Annual Workshop on Comp. Learning Th., pp. 128–136. Morgan Kaufmann, San Francisco (1991)
11. Drewes, F., Högberg, J.: Query Learning of Regular Tree Languages: How to Avoid Dead States. Theory of Computing Systems 40(2), 163–185 (2007)

# On Learning Regular Expressions and Patterns Via Membership and Correction Queries

Efim Kinber

Department of Computer Science, Sacred Heart University, Fairfield, CT 06825-1000, U.S.A.
kinbere@sacredheart.edu

**Abstract.** Based on the ideas suggested in [5], the following model for learning from a variant of *correction* queries to an oracle is proposed: being asked a membership query, the oracle, in the case of negative answer, returns also a *correction* – a positive datum (that has not been seen in the learning process yet) with the smallest edit distance from the queried string. Polynomial-time algorithms for learning a class of regular expressions from one such query and membership queries and learning pattern languages from queries of this type are proposed and discussed.

## 1 Introduction

In this paper, we propose and discuss some algorithms for learning pattern languages and a class of regular expressions from positive data. There exist different models for learning languages from examples, in particular, the popular Gold's model [9] for learning languages in the limit from a stream of all positive examples, Angluin's model [3,4] for learning languages from queries to a teacher (oracle), and Valiant's PAC learning model [18]. In this paper, we follow the Angluin's query model representing learning process as an interactive session between a learner and a teacher. Specifically, the learner asks the teacher queries of a certain type, and the teacher returns correct answers to the queries. After a finite number of queries, the learner must output a correct description of the target language. This description of the target language must be within certain general class chosen by the teacher and known to the learner (for example, regular expressions, patterns, DFAs, an acceptable numbering of all recursively enumerable sets, etc.)

Over the years, capabilities of different types of queries for various classes of target languages have been explored within the framework of the Angluin's query model. In particular, in her seminal paper [3], D. Angluin presented a polynomial-time algorithm using membership and equivalence queries and learning (minimal) deterministic finite automata. Since then, query model has been used for studies of learning various classes of target concepts via membership, equivalence, subset, superset and disjointness queries. Among the classes of target concepts, context-free grammars ([14]), non-deterministic finite automata ([19]), regular tree languages ([17]), and some others have been used.

Using the general framework of the Angluin's query model, L. Becerra-Bonache, A. H. Dediu, and C. Tîrnăucă introduced in [5] a new type of queries - so called *correction queries*. A correction query is a modification of the most popular and most natural type of queries defined by D. Angluin - membership queries. While in response to a membership query, a teacher just answers 'yes' or 'no', responding to a correction query, the teacher, in case the answer is negative (and, thus, the queried string is not in the target language), provides also a shortest extension of the queried example (string) belonging to the target language. This approach to modeling learning processes stems from the following observation discussed, in particular, in [13]: while learning a language, in addition to *overt* explicit negative evidence (when a parent points out that a certain statement by a child is grammatically incorrect), a child often receives also *covert* explicit evidence in form of corrected or rephrased utterances. The choice of correction in form of a shortest extension of a wrong queried string in [5] and some other works following this line of research (e.g. [16]) has been dictated by the nature of classes of languages being learned. For example, C. Tîrnăucă and T. Knuutila in [16] consider learning so-called $k$-reversible regular languages (introduced and shown to be learnable in the limit within the Gold's learning model by D. Angluin in [2]). For these languages, where a prefix of a string in a language defines the set of all extensions belonging to the language, an extension of an incorrect string seems to be the most natural type of a "closest" correction. They also apply this model to learning pattern languages. In [6], the authors propose a different type of corrections: the correction string is "closest" to the queried one in terms of shortest *edit distance*; they do not impose any other restrictions on the correction string.

In this paper, we consider learning the class of patterns (introduced by D. Angluin in [1]) and a class of regular expressions using membership queries and a type of correction queries similar, but somewhat different from the one in [6], which is more suitable to the nature of main operations forming strings in the corresponding target languages. Namely, for our class of regular expressions, the main operation forming strings in corresponding languages will be looping, while in patterns such operation is substitution of variables by strings. Accordingly, for the concept classes under consideration in this paper, we consider the following type of correction queries: if the queried string is not in the target language, then the teacher returns the positive example with a smallest edit distance from the queried string and previously not used in the learning process (the requirement of correction being not previously used in the learning process is not a part of the learning model in [6]). For our class of regular expressions, we use one such query to get the shortest string in the target language, and then use membership queries. For learning patterns, we use correction queries only. Within this framework, we suggest polynomial-time algorithms for learning respective classes of concepts.

Learning pattern languages has been a subject of intensive study since the introduction of this concept by D. Angluin in 1979. A comprehensive survey of results in this area, including some interesting practical applications, can be found in [15].

While there is a significant body of work on polynomial-time learning of deterministic finite automata, relatively little has been done on learning regular expressions (practically all articles on this subject are listed in [10]), whereas regular expressions are often more suitable for specifying regular languages by human beings than, say, DFAs ([10]) and more suitable for representing various learning tasks (see, for example, [7] and [8] where learning algorithms for regular expressions have been designed for inference of Document Type Descriptions and XML Schema Definitions). Thus, we hope that this work will contribute to a better understanding of how regular expressions can be efficiently learned.

## 2 Notation and Preliminaries

Let $\Sigma$ be a finite alphabet. $\Sigma^*$ denotes the set of all finite strings (words) over $\Sigma$. A *language* is any subset of $\Sigma^*$. The length of a string $w$ is denoted by $|w|$. $uv$ denotes the concatenation of the strings $u$ and $v$. Let $\epsilon$ be a character not belonging to $\Sigma$. Given any two strings $u$ and $v$, the *Levenshtein edit distance* $d(u, v)$ is defined as the minimum number of operations needed to transform one string to another, where an operation is insertion, deletion, or substitution of a single character.

By a substring of a string $w$, we will understand any string $u$ such that $w = vux$ for some strings $v$ and $x$. For any string $v$ and any number $k$, let $v^k$ denote the concatenation of $k$ copies of $v$.

Let $w = a_1a_2 \dots a_k$ be a string. Any string $a_ia_{i+1} \dots a_ka_1a_2 \dots a_{i-1}$ is called a *circular shift* of $w$. For any string $v$, the regular expression $(v)^+$ will be called a *loop*, and the string $v$ will be called the *body* of this loop.

## 3 Learning Via Queries

The classes of languages considered in this paper are examples of so-called *indexable* classes of recursive languages over $\Sigma^*$. A class of recursive languages $\mathcal{L}$ is an *indexable* class if there exists an effective numbering $L_i, i = 0, 1, 2, 3, \dots$ of all the languages in $\mathcal{L}$ such that membership in these languages is uniformly decidable (in other words, there is a recursive function that, for any $w \in \Sigma^*$, and any index $i$, outputs 1 if $w \in L_i$ and 0, otherwise). We consider the following model of learning of an indexed class $\mathcal{L}$: a learner $M$ is an algorithmic device that has access to an oracle that truthfully answers queries of a certain type. Having received an answer for its query, the learner $M$ either generates a new query to the oracle, or returns a conjecture, an index of a language in $\mathcal{L}$, and halts. If the conjecture, say $i$, is an index of the target language $L$, then we say that $M$ *learns* the target language $L$. We say that $M$ learns the class $\mathcal{L}$ if $M$ learns every language in this class. Obviously, for every language in a learnable class, the learner $M$ asks a finite number of queries.

In this paper, we will consider the following types of queries:

*Membership queries.* The learner asks if a string $w$ belongs to the target language and gets the answer "yes" or "no".

*Correction queries.* The learner asks if a string $w$ belongs to the target language $L$. If the answer is negative, the teacher (oracle) returns also a *correction* - a string $v \in L \backslash A$ such that

$$|v| = |w| \text{ and } d(v, w) = min\{d(u, w) | u \in L, u \notin A\}$$

where $A$ is the (finite) set of all strings for which the learner has received the answer "yes" on the previous steps and all the correction strings received by the learner on the previous steps (in other words, the oracle never returns positive examples of the target language seen on the previous steps of the learning process). If no such string $v$ exists, the teacher returns a shortest string of the length different from $|w|$ such that $d(v, w) = min\{d(u, w)|u \in L$ and $u \notin A\}$. (In other words, preference is always given to correction strings of the same length, if any). If no such string $v$ exists, the teacher returns $\epsilon \notin \Sigma$. We will also assume that, among the correction strings of the same length, the teacher always chooses the smallest one in the lexicographic order (for patterns, we will consider learning via correction queries for the languages over the binary alphabet $\Sigma = \{0, 1\}$ only, and will assume that $0 < 1$, and empty character $< 0$).

The learning algorithms presented in the next two sections will have running times polynomial in the length of the shortest string in the target language (or, in the length of the target expression).

## 4 Learning a Class of Regular Expressions

We consider the following regular expressions over the alphabet $\Sigma$:

$$u_1(v_1)^+ u_2(v_2)^+ \dots u_n(v_n)^+ u_{n+1},$$

where $u_i, v_i, i = 1, 2, \dots, n$ are strings over $\Sigma$ (at least one of them must be nonempty). Thus, all loops must be used at least once, unions are not used, and the loop depth is 1 (similar regular expressions for a different learning task were considered in [11]).

An example of such an expression is:

$$a(aa)^+ bccd(cd)^+ ddabb^+ b(abb)^+.$$

A regular expression $\alpha$ in our class is called *left-aligned* if, for any string $v \in \Sigma^+$, there are no segments $v^{m_1}(u_1)^+ v^{m_2}(u_2)^+ v^{m_k}(u_k)^+$ in $\alpha$, where each $u_i, i = 1, 2, \dots, k$ is $v^r$ for some $r$, and at least one $m_j > 1$. For instance, the above example of the regular expression is not left-aligned, and it is equivalent to the (unique) left-aligned expression $(aa)^+ abc(cd)^+ cddab^+ bb(abb)^+$.

In other words, in left-aligned expressions, all loops are shifted to the left as much as possible. We will limit expressions in our class to just left-aligned expressions. We will also assume that if, in an expression $\alpha$ in our class, there is a subexpression-loop $\beta = (v^k)^+$ for some (shortest possible) string $v$ and some $k$, then there is no $(v^m)^+$ neighboring $\beta$ on the left or on the right in $\alpha$. For example, subexpressions $((ab)^3)^+((ab)^5)^+$ are not allowed.

Let $R1^+$ denote the class of regular expressions satisfying the above conditions. For any expression $\alpha$ in this class, let $L(\alpha)$ denote language defined by $\alpha$. We will also use $R1^+$ to denote the class of corresponding languages.

We will often use shifting of loops in regular expressions. Sometimes, in our algorithm, we will use shifting individual loops to the right as much as possible, right-aligning them (and, obviously, preserving equivalence). For example, in the left-aligned expression $(a)^+a(ab)^+ababa(b)^+b$, right-aligning the loop $(ab)^+$ will result in the expression $(a)^+aababab(ab)^+b^+$ (note that the loop $(b)^+$ has also been shifted to the right as the result of right-aligning the loop in question).

## 4.1 The Learning Algorithm for $R1^+$

We now present an algorithm that learns any language in $R1^+$ using one correction query and membership queries. The algorithm will always output a (unique) left-aligned expression representing the target language.

To make our presentation of the algorithm clear, we will begin with an example (the example uses a two-letter alphabet $\Sigma$, however, the algorithm works for expressions over arbitrary finite $\Sigma$). Consider the following (left-aligned) expression in $R1^+$:

$$a^+a(ab)^+ab^+(ab)^+ab^+a(bb)^+.$$

Let $L$ denote the target language. On the very first (special) step of the algorithm, ask the correction query for the empty string. Obviously, the teacher will return the shortest string $w = aaababababababb$. Now we switch to the main part of the algorithm. On its first phase, the algorithm tries to find all one-letter loops. It finds all the longest one-letter substrings, listing them in the order they are found in the string $w$ - in our case, they are $aaa, b, a, b, a, b, a, b, a, bb$. Then, for each block, it determines if there are loops associated with it. First, query '$aaaababababababb \in L$?' (using one extra $a$ in the first block). As the answer is 'yes', transform $w$ to $a^+aababababababb$. Using similar queries, the algorithm arrives to to the expression $r_1 = a^+aabab^+abab^+abb$.

Now the algorithm attempts to find all two and more-letter loops. First, the algorithm finds all two-letter loops. As we already have the loop $a^+$, the algorithm does not attempt to construct the loop $(aa)^+$. Thus, the algorithm will try all three strings $ab$ and the tail $bb$ of $r_1$ . First, it will query '$aaabababbabababb \in L$?'. Obviously, the answer is 'yes'. Note that the first substring $abb$ in the queried string can be contributed only by (already found) subexpression $ab^+$, and, thus, the first substring $abab$ must have been contributed by the loop $(ab)^+$ *before* the subexpression $ab^+$. Note also that the (obvious) query '$aaabababababababb \in L$?' (simply using the first substring $ab$ twice) would not work, as the substring $abababab$ after the prefix $aa$ could have been contributed by the subexpression $abab^+(ab)^+$ containing the loop $(ab)^+$ *after* the subexpression $ab^+$. Given the answer 'yes', the algorithm conjectures the expression $a^+a(ab)^+ab^+abab^+abb$, and queries '$aaababbabababbabb \in L$?' to find out if the substring $ab$ after the first loop $b^+$ is the body of a loop. The answer is 'yes'. Note that, again, the first and the second substrings $abb$ could have been contributed only by the (already found) subexpressions $ab^+$, and, thus, the substring $abab$ has been contributed by the loop $(ab)^+$ between them. Thus, our next (intermediate) conjecture is $a^+a(ab)^+ab^+(ab)^+ab^+abb$. Now, using the query '$aabababababbababb \in L$?', the algorithm will try to determine if the last substring $ab$ in $w$ is the body of a loop. The answer is negative, and the last query '$aaabababababbbb \in L$'? will find the last two-letter loop $(bb)^+$. Then the algorithm, in a similar way, will determine that there are no three- or more letter loops, and output the correct expression.

Now we present a formal description of the learning algorithm.

**Algorithm**

Let $L$ denote the target language.

On the Phase 1, query empty string. Let $w$ be the correction string (obviously, the shortest one in the target language). Now, for any substring $v$ in $w$, let $w_l(v)$ and $w_r(v)$ denote the strings such that $w = w_l(v)vw_r(v)$.

PHASE 2 (It uses membership queries only):

On the STEP 1, inserting one extra character $a$ at the end of each (maximal) substring $aa \dots a$ in $w$ (same for $b$), and making a query for the corresponding string (of the length $|w| + 1$), if the answer is 'yes', replace the first character of the substring by the loop $(a)^+$ (keeping the resulting exression, denoted $r_1$, left-aligned).

If there are no substrings of the type $ab$ (after, possibly, shifting a loop $(b)^+$ to the right) in $r_1$, then terminate and output $r_1$ as the target expression. Otherwise, go to STEP 2.

STEP $n$: Let $r = r_{n-1}$ be the regular expression obtained on the step $n - 1$. By induction, we assume that $r$ is left-aligned. Let

$$r = (\alpha_1)^+\beta_1(\alpha_2)^+\beta_2 \dots (\alpha_k)^+\beta_k$$

for some $k$ and some $\alpha, \beta \in \Sigma^*$.

On each iteration $j$ of the FOR loop below, the algorithm will be constructing loops between (possibly shifted to the right on the previous iteration) the loop $\alpha_j^+$ and the right-aligned loop $\alpha_{j+1}^+$. Let $r^0 = r$.

```
        FOR $j = 1, 2, \dots k$:

Step $j$:
  Let $t$ be the expression obtained from $r^{j-1}$ (the expression from $j-1$ iteration
     of the FOR-loop) and by right-aligning $\alpha_{j+1}^+$ as much as possible.
  Let $u$ be the substring in $t$ between the loop preceding $\alpha_{j+1}^+$ and $\alpha_{j+1}^+$. If
     $|u| < n$ (thus, there is not enough space for another loop between two
     neighboring loops), then let $r^j = r^{j-1}$ and go to the next iteration of the
     FOR loop. Otherwise, let $u = a_1a_2 \dots a_p$ for $a_1, a_2, \dots, a_p \in \Sigma$.
  In the WHILE loop below, the algorithm will attempt to construct loops
     based on the substrings in $u$ (with the length of the body $n$), thus trans-
     forming $t$. Let $Tail = u$.
  WHILE ($|Tail| \geq n$) DO
       Let $x$ be the prefix of $Tail$ with the length $|x| = n$. Let $x = v^i$ for
          some (shortest) $v$. If $x$ is preceded in $t$ by the loop $(v^k)^+$ for some $k$,
            then remove from the $Tail$ the shortest prefix such that the prefix of the length $n$ of the remaining string is neither $x$ nor $(z)^i$ for some circular shift $z$ of $v$. Let $Tail$ be the remaining string. (For example, if $n = 4$, $Tail$ is $ababababcccc$ and preceded in $t$ by $(ab)^+$, then $Tail$ is set to $babcccc$; if $ababababacccc$ is preceded by the same loop, then $Tail$ is set to $abacccc$). Go to the top of the WHILE loop.
        If $x$ is not preceded by any such $(v^k)^+$, then the goal is to determine if $x$ is the body of a loop in the target expression. Let $z$ be the longest extension of $x$ in $Tail$ equal to $v^k$ for some $k$.
        Consider all loops $(\gamma)^+$ to the right from $z$ in the expression $t$. Let $t'$ denote the expression obtained from $t$ by substituting each loop $(\gamma)^+$ by the string $\gamma$. Let $z'$ be the longest extension of $z$ in $t'$, of which $z$ is a prefix, such that $z' = v^m$ for some $m$. Now substitute all loops $(\delta)^+$ to the left from $Tail$ in $t$ by $\delta$, and let $t''$ be the corresponding expression obtained from $t$. Let $z''$ be the longest substring in $t''$ extending $Tail$ to the left such that $z'' = v^m$ for some $m$.
        Now the following cases are possible:
        Case 1: $z' = z$ and $z'' = z$. Query '$w_l(x)xxw_r(x) \in L$?'.
        Case 2: $|z'| > |z|$ and $z'' = z$. Then, there is a substring $\gamma$ in $w_r(x)$, which is the body of the leftmost loop $(\gamma)^+$ mentioned above. Replace this $\gamma$ in $w_r(x)$ by $\gamma\gamma$, and let $w'$ be the string obtained from $w_r(x)$. Query '$w_l(x)xxw' \in L$?'.
        Case 3: $z' = z$ and $|z''| > |z|$. Then, there is a substring $\delta$ in $w_l(x)$ which is the body of the rightmost loop $(\delta)^+$ mentioned above. Replace this $\delta$ in $w_l(x)$ by $\delta\delta$, and let $w''$ be the string obtained from $w_l(x)$. Query '$w''xxw_r(x) \in L$?'.
        Case 4: $|z'| > |z|$ and $|z''| > |z|$. Then, as in cases 2 and 3, replace $\gamma$ by $\gamma\gamma$ in $w_r(x)$, getting $w'$, and replace $\delta$ by $\delta\delta$ in $w_l(x)$, getting $w''$. Now query '$w''xxw' \in L$?'.
        In all four cases, if the answer is 'yes', substitute $x$ in $t$ by $(x)^+$, remove the prefix $x$ from $Tail$, and go to the top of the loop. If the answer is 'no', then remove from $Tail$ the longest prefix $(x)^i y$, where $x = yz$ for some $y$ and $z$, so that the prefix of the remaining string is not $zy$ (for example, if $x = abb$ and $Tail$ is $abbabbabbaccc$, then the $Tail$ becomes $baccc$, rather than $bbaccc$); go to the top of WHILE loop.
    EndWhile
Left-align all loops in the current expression $t$. Set $r^j = t$. End step $j$.

  END STEP $n$.
  Return $r_k$ for $k = |w|$ as the target expression.
```

## 4.2 Correctness of the Algorithm

Correctness of the algorithm is proved by induction. By induction, let us assume that all loops that have been created before some iteration of the WHILE loop on Step $j$ of the FOR loop on STEP $n$ are correct. Now, we will show that, on the given interation of the WHILE loop, either a new loop cannot be created, or, if it is created, it is correct. Consider the prefix $x$ of $Tail$ of the length $n$ (as defined in the WHILE loop). As defined in the WHILE loop, let $x = v^i$ for the shortest possible $v$ and some $i$. Consider the case when the loop to the left from $x$ in the current expression $t$ is $(v^k)^+$ for some $k$. Then $x$ cannot be the body of a loop in the target expression, and $Tail$ is reduced appropriately (so that copies of $x$ or its circular shifts following the given $x$ would not be tested as bodies of possible loops later). In all four Cases (as defined in the WHILE loop), if the answer is 'no', then $x$ obviously cannot be the body of a loop. Now, we will assume that the answer is 'yes'. In the *Case 1*, the second $x$ in the query can be contributed to the string only by the loop corresponding to the first $x$ (as the target expression is left-aligned). Now consider the remaining three cases and the strings $\gamma$ and $\delta$ as defined in the WHILE loop. In the *Case 2*, $\gamma$ (the body of the leftmost loop to the right from $x$) cannot be equal to any $v^e$ or $y^e$ for some circular shift $y$ of $v$, as, otherwise, the corresponding loop would have been shifted to the left from $x$ (note that every expression $t$ in the loop WHILE is left-aligned). Thus, neither the string with the prefix $x$ obtained when $u$ is extended by substituting $\gamma$ by $\gamma\gamma$, nor any its extension to the right in $t$ can be equal to $v^p$ for any $p$. Therefore, if the answer to the query in *Case 2* using $xx$ is 'yes', the second $x$ can be contributed by neither the loop $(\gamma)^+$, nor a loop to the right from it in the target expression. Now, in the *Case 3*, $\delta$ (the body of the rightmost loop to the left from $x$) cannot be equal to $v^e$ (or its cicular shift), since, otherwise, $x$ could not be a prefix of $Tail$. Thus, if the answer to the query in this case is 'yes', the second $x$ cannot be contributed by the loop $(\delta)^+$ or a loop to the left from it, and, similarly, by neither $(\gamma)^+$, nor $(\delta)^+$, nor loops to the right from the former, or to the left from the latter, respectively, in the *Case 4*.

### 4.3 Complexity of the Learning Algorithm

The total number of queries asked is obviously $O(n^3)$, as on each step of the WHILE loop, the algorithm makes just one query. On each step of the WHILE loop, the algorithm performs some work that requires time $O(n^2)$. Thus, overall complexity of the algorithm is $O(n^5)$.

### 4.4 Modifications of the Class $R1^+$

It would be interesting to explore if some modifications of the class $R1^+$ were learnable in polynomial time. One such modification is the class $R1$ that contains loops $(v)^*$ rather than $(v)^+$. However, this class may be too hard to learn in polynomial time. A modification of this class, $R2$, would contain only those loops $(v)^*$ where the body $v$ did not contain any repetitions (for example, $v = abb$ would not be allowed). We can exhibit a polynomial-time learning algorithm using one correction query and membership queries for the following limited version $R2_1$ of $R2$: one-letter loops are allowed only, and there is a nonempty string between any two loops. An example of such a regular expression is $a^*aab^*ac^*ca^*aa$. $R2_1$ is a subclass of the class of regular expressions considered in [10] (where unions are allowed), however, the algorithm in [10] learns regular expressions in the limit, while in our model, the first conjecture must be correct. We omit details due to limitations on the size of the paper.

Another approach to learning modifications of $R1^+$ containing loops $(v)^*$ would involve variants of correction queries, or both membership and correction queries.

In, probably, the most recent paper on learing regular expressions [8], the authors define an interesting (and practically useful) class of regular exressions and design polyomial-time algorithms for inference of the expressions in this class from positive data. All expressions in [8] must be *deterministic* (or *one-unambiguous*) and cannot have more than a uniformly bounded number of occurrences of each alphabet symbol. Our class $R1^+$ does not have these restrictions.

## 5 Learning Pattern Languages Via Correction Queries

Learning patterns has been a subject of intensive study since their introduction by D. Angluin in [1]. A *pattern* $\pi$ over a finite (in our case - binary) alphabet $\Sigma$ and a countable infinite set $X = \{x_1, x_2, \ldots\}$ of variables is a (nonempty) string in $(\Sigma \cup X)^*$. An example of a pattern is $x_1x_1x_201x_1x_30$. The (non-erasable) *pattern language* $L(\pi)$ consists of all strings obtained by substituting the variables in the pattern $\pi$ by arbitrary strings in $\Sigma^+$. For example, substituting $x_1$ by 01, $x_2$ by 00, and $x_3$ by 1 in the above example, we get the string 010100010110.

Different authors used different paradigms of learning to study learnability of the class of pattern languages. Among the latest works on this topic, is the paper [16], where patterns are being efficiently learnt from correction queries, and, in case of the answer 'no', the teacher returns the shortest *extension* $u$ of the queried string $v$ belonging to the target language. Unlike their approach, our algorithm for learning pattern languages uses the type of queries introduced in Section 3.

### 5.1 The Learning Algorithm

Let $L$ be the target language. First, we present our algorithm on two examples. Our first example is the pattern $\pi = x_101x_2x_3x_3x_3x_3x_3x_3x_3x_4x_4x_4x_5x_5x_5$. Let $\pi(r), 1 \leq r \leq |\pi|$, denote the $r$-th character in $\pi$ (either a variable, or a constant).

First, query the empty string. The oracle will obviously return the correction string $w = 0010^{14}$ (of total length 17). Now query '$1^{17} \in L$?' . The answer is 'no', and the oracle returns the correction string $101^{15}$. Now we know that $\pi(2) = 0, \pi(3) = 1$, and all other $\pi(r)$ are variables. Our goal now is to find these variables. First, query '$1010^{14} \in L$?', trying 1 for the first position. The answer is 'yes'. The algorithm sets $\pi(1) = x_1$. Now, query '$00110^{13} \in L$?', trying 1 for the fourth position. The answer is 'yes', and the algorithm sets $\pi(4) = x_2$. Now, query '$001010^{12} \in L$?', trying 1 for the 5-th position. The answer is 'no', and the correction string is $10110^{13}$ (note that strings $1010^{14}$ and $00110^{13}$ are closer to the queried string, however, both of them have already been used). Now query '$101110^{13} \in L$?', trying 1 for the 5-th position again.

The answer is 'no' again, and this time the (smallest in lexicographic order) correction string is $10110^{10}111$. As the tail 111 has never appeared, the algorithm sets $\pi(15) = \pi(16) = \pi(17) = x_3$ (the numbers of variables in the output of the algorithm can obviously be different from the ones in the original pattern). Now, query '$101110^9111 \in L$?', trying 1 for the 5-th position once again. This time the correction string is 10111111111000111, and the algorithm sets $\pi(r) = x_4$ for $5 \le r \le 11$. Now, it queries '$0010^8100000 \in L$?', trying 1 for the 12-th position. The answer is 'no' and the correction string is $0010^8111000$. The algorithm sets $\pi(12) = \pi(13) = \pi(14) = x_5$. As all variables have been found, the algorithm returns $\pi$ as the target pattern.

Our next example is the pattern $\pi = 10x_1x_1$. Using first two queries (and getting correction strings 1000 and, respectively, 1011), the algorithm will find constant characters 1 and 0. Then it will query '$1010 \in L$?'. The answer is 'no', but the oracle must return a string $u$ with the length $|u| > 4$, as all the strings in $L$ of the length 4 (in particular, 1011) have already been seen. In this case, the algorithm sets $\pi(r) = x_1$ for $r \in \{3, 4\}$ and terminates.

Now we give a description of the learning algorithm.

**Algorithm**

Query if the empty string is in $L$. The teacher will return a string $w = a_1a_2 \dots a_n$. Note that all characters 1 in this string must be constants, while all values 0 are either constants, or correspond to occurrences of variables, as the string $w$ must be the smallest in the lexicographic order among all the shortest strings (of the length $n$) in the target language. If there are no 0 in $w$, the algorithm returns the result $1^n$ and terminates.

Otherwise, first consider the special case of $w = 0$. Query '$1 \in L$?'. If 'yes' then set $\pi = x_1$, if 'no', set $\pi = 0$ and terminate the algorithm.

Now consider the case $|w| > 1$. Our goal is to determine which characters $a_i$ must be replaced by variables, and which are constants 0 (constants 1 have already been determined). Query '$1^n \in L$?'. If the answer is 'no' (thus, the target pattern contains some constants 0) , let $u = b_1b_2 \dots b_n$ be the correction string returned (note that it will have the same length $n$ as the queried string). If the answer is 'yes', let $b_r = 1, 1 \le r \le n$.

Now the algorithm enters the FOR loop executed for $j = 1, 2, \dots n$. In this loop, let $\pi_i$ denote the pattern output at the end of the step $i$. In $\pi_i$, let $\pi_i(j)$ denote the symbol on the $j$-th position in $\pi_i$ (either a variable, or a constant). For all $j \in \{1, 2, \dots, n\}$, we set $\pi_0(j) = 1$ if $a_j = 1$, and $\pi_0(j) = 0$, otherwise.

Now we describe the step $j$ of the loop. Let $\pi = \pi_{j-1}$. If either $\pi(j) = 1$, or $\pi(j) = b_j = 0$, or $\pi(j)$ is a variable, then the corresponding symbol in the target pattern is a constant or an already found variable. Thus, set $\pi_j = \pi$ and go to the step $j + 1$ of the loop (or terminate if $j = n$).

Otherwise, we have $\pi(j) = 0$ and $b_j = 1$. This means that there must be a variable on the position $j$ in the target pattern. If all symbols $\pi(r)$ for $r > j$ are already variables or constants 1, then just set $\pi_j(j)$ to a new variable, set $\pi_j(r) = \pi(r)$ for all $r \ne j$ and terminate the loop. Otherwise, let $\alpha$ be the string obtained from $\pi$ by substituting $\pi(j)$ by 1 and all occurrences of variables by 0.

Query '$\alpha \in L$?' (note that $\alpha$ contains at least one symbol 0 on a position $s > j$ where $b_s = 1$, and, thus, it is not $u$ or $1^n$ seen before). If the answer is 'yes', then set $\pi_j(j)$ to a new variable, set $\pi_j(r) = \pi(r)$ for all other $r$ and go to step $j + 1$.

If the answer is 'no' (this means that there must be other occurrences of the same variable as on the position $j$ in the target pattern), then let $\beta$ be the correction string. If $|\beta| > n$ (thus, all corrections of the length $n$ have already been seen), then set $\pi_j(j)$ and all other $\pi_j(r)$ for $r$ such that $\pi(r) = 0$ and $b_r = 1$ to a new variable (same for all such $r$) and terminate the algorithm.

Otherwise, let $\beta = c_1c_2 \ldots c_n$. The algorithm enters the following WHILE loop that runs until $c_j$ in the last correction string $\beta$ becomes 1. On each step of this loop,

(1) set all $\pi_j(r)$ for $r$ such that $\pi(r) = 0$ and $c_r = 1$ to a new variable; if $c_j = 1$, terminate the WHILE loop.

(2) replace $c_j$ by 1 in $\beta$, and let $\gamma$ be the modified string; query '$\gamma \in L$?' (note that the answer is 'no' - otherwise, the answer 'yes' would have been given earlier). Let $\beta$ be the correction string. If $|\beta| > n$, then, as in the corresponding case above, substitute all $\pi(r) = 0$ for which $b_r = 1$ by a new variable and terminate the algorithm. Otherwise, let $\beta = c_1c_2 \ldots c_n$. Return to the top of WHILE loop.

Once the loop WHILE has terminated, go to step $j + 1$ of the FOR loop.

Return the pattern $\pi_j$ created on the last executed STEP $j$ of the FOR loop.

## 5.2 Correctness of the Algorithm

By induction on $j$, let us assume that the initial segment $\pi_{j-1}(r), r \leq j - 1$, of $\pi_{j-1}$ and all variables and constants 1 among $\pi_{j-1}(r)$ for $r > j$ of the target pattern being constructed, have been constructed correctly. Now, we will show that the same is true for $\pi_j$ .

Obviously, the only interesting case is when $\alpha$ is queried as described in the algorithm. Note that this $\alpha$ has not been used yet in the learning process: the only strings used before in the learning process and having 1 on the position $j$ were $1^n$ and, possibly, the correction string $u$ (containing 1-s for all variables), however, as it is pointed out in the description of the algorithm, $\alpha$ is neither $u$, nor $1^n$. If the answer is 'yes', then substituting just $\pi_{j-1}(j) = 0$ by the single occurrence of a new variable is obviously correct. Now suppose the answer is 'no'. Then there must be some new variable (not used before), say $x$, on the position $j$, however, there are some other occurrences of this variable in the target pattern, and we don't know them yet. Now, suppose the correction string $\beta$ has the length greater than $n$. If some 0 on a position $r > j$ in $\pi_{j-1}$ for which $b_r = 1$ is not a value of $x$, then the oracle should have returned some string $\beta \in L$ with 1 on the position $j$ and 0 on the position $r$, as such a string has not been seen yet. However, it returned a longer string. Therefore, all values 0 on the positions $> j$ in $\pi_{j-1}$ where $b_r = 1$ are values of the same variable $x$ and, thus, $\pi_j$ is the correct (final) target pattern.

If $|\beta| \leq n$, then the algorithm enters the WHILE loop. In the case (1), the algorithm observes the correction string, where some already existing variables have been replaced by 1, and, possibly 0-s on the positions $> j$ in $\pi$; as the correction string is at the shortest distance from the queried one, all these replaced 0-s (if any) must be the values of the same variable. Thus, the algorithm is obviously correct, having replaced all $c_r$ in question by occurrences of a new variable.

In the case (2), if $|\beta| > n$, then the analysis is as above. Otherwise, the algorithm returns to the case (1) with the new correction string.

Now note that, on some iteration of the WHILE loop, the oracle must return either a correction string $\beta$ with $|\beta| = n$ and $c_j = 1$ (as $c_j$ is a value of a variable, and, since oracle each time returns a string of the length $n$ not seen yet, some correction string of the length $n$ must have $c_j = 1$), or with $|\beta| > n$. Thus WHILE loop always terminates, whenever entered.

Therefore, the algorithm correctly learns the target pattern $\pi$.

## 5.3 Complexity

The WHILE loop runs at most $O(n)$ times, as every correction string contains more characters 1 than the one on the previous step. Creating (possibly) a new vaiable in the body of this loop requires time $O(n)$. Thus, the total running time of the algorithm comes to at most $O(n^3)$. The total number of queries in the WHILE loop is $O(n)$, and, thus, the total number of queries is $O(n^2)$.

## 5.4 Discussion

Two distinctive features of the oracle in our model are that a) it returns a correction not previously seen, and b) it gives preference to corrections of the size equal to the size of the queried string. Both of them are important for the success of our algorithm. If the oracle may return previously seen corrections, then, in many cases, it will keep providing the string that has been utilized, and the learner will not be able to get necessary information about unknown variables. Also, corrections of arbitrary length (even closest to the queried string) are of little help (if a returned string is longer than the pattern being constructed, our algorithm just uses this fact to finish its work, but not the correction string itself). The fact that positive examples of arbitary length are not helpful for learning patterns when the shortest string in the target language becomes available, was noticed (and successfully utilized) yet in [12], where the learning algorithm, while learning patterns in the limit from all positive data, simply ignores all examples longer than the shortest positive example seen so far. Our algorithm is similar to the algorithm in [12] in this respect - however, it gets (and, accordingly, utilizes) input data differently.

It is possible to slightly modify our query model, leaving the learning algorithm intact. Instead of using requirement of selecting lexicographically smallest correction string, we could use correction strings that contain largest number of 0-s among the ones at the shortest edit distance from the queried string (still giving preference to correction strings of the length equal to the length of the queried string).

As it was shown in [16], patterns cannot be learned in polynomial time using membership queries. Thus, corrections are essential for polynomial-time learnability.

There are multiple ways of relaxing constraints of our query model. First, within the framework of the given model, it would be interesting to find out if polynomial-time learnability can be preserved for patterns over arbitrary alphabets $\Sigma$. Another interesting question is if the requirement of preference given to correction strings smallest in the lexicographic order (or containing largest number of 0-s) can be lifted. Yet another interesting case would be when a correction string were just at the shortest edit distance from the queried one - without any other constraints. However, we conjecture that a polynomial time algorithm could not exist in this case. One more interesting open problem is to find out if polynomial-time learning of patterns is possible while dropping the requirement that correction string must be selected among those not seen so far in the learning process (in this case, selection of the correction string will not depend on the learning algorithm, and the teacher will not need to remember which strings have been used in the learning process). Yet another interesting open problem is whether, within the framework of our model, polynomial-time algorithms exist for patterns over alphabets of the size 3 or more.

**Acknowledgments.** The author is grateful to C. Tîrnăucă for a useful discussion and to anonymous referees for several helpful comments and suggestions.

## References

1. Angluin, D.: Finding Patterns Common to a Set of Strings (extended abstract). In: 11th Annual ACM Symposium on Theory of Computing, pp. 130–141. ACM Press, New York (1979)
2. Angluin, D.: Inference of Reversible Languages. Journal of the ACM 29(3), 741–765 (1982)
3. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Information and Computation 75(2), 87–106 (1987)
4. Angluin, D.: Queries and Concept Learning. Machine Learning 2, 319–342 (1988)
5. Becerra-Bonache, L., Dediu, A.H., Tîrnăucă, C.: Learning DFA from Correction and Equivalence Queries. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 281–292. Springer, Heidelberg (2006)
6. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Learning Balls of Strings with Correction Queries. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 18–29. Springer, Heidelberg (2007)
7. Bex, G.J., Neven, F., Schwentick, T., Tuyls, K.: Inference of Concise DTDs from XML Data. In: 32nd International Conference on Very Large Data Bases VLDB (2006)
8. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. In: WWW Conference 2008, Beijing, China, pp. 825–836 (2008)
9. Gold, E.M.: Language Identification in the Limit. Information and Control 10, 447–474 (1967)

10. Fernau, H.: Algorithms for Learning Regular Expressions. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 297–311. Springer, Heidelberg (2005)
11. Kinber, E.: Learning a Class of Regular Expressions via Restricted Subset Queries. In: Jantke, K.P. (ed.) AII 1992. LNCS, vol. 642, pp. 232–243. Springer, Heidelberg (1992)
12. Lange, S., Wiehagen, R.: Polynomial-time Inference of Arbitrary Pattern Languages. New Generation Computing 8(4), 361–370 (1991)
13. Rohde, D.L.T., Plaut, D.C.: Language Acquisition in the Absence of Explicit Negative Evidence: How Important Is Starting Small? Cognition 72, 67–109 (1999)
14. Sakakibara, Y.: Learning Context-free Grammars from Structural Data in Polynomial Time. Theoretical Computer Science 76, 223–242 (1990)
15. Shinohara, T., Arikawa, S.: Pattern Inference. In: Lange, S., Jantke, K.P. (eds.) GOSLER 1994. LNCS, vol. 961, pp. 259–291. Springer, Heidelberg (1995)
16. Tîrnăucă, C., Knuutila, T.: Polynomial Time Algorithms for Learning k-reversible Languages and Pattern Languages with Correction Queries. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 264–276. Springer, Heidelberg (2007)
17. Tîrnăucă, C.I., Tîrnăucă, C.: Learning Regular Tree Languages from Correction and Equivalence Queries. Journal of Automata, Languages and Combinatorics 12 (2007)
18. Valiant, L.G.: A Theory of the Learnable. Communications of the ACM 27(11), 1134–1142 (1984)
19. Yokomori, T.: Learning Non-deterministic Finite Automata from Queries and Counterexamples. Machine Intelligence 13, 169–189 (1994)

# State-Merging DFA Induction Algorithms with Mandatory Merge Constraints

Bernard Lambeau[1], Christophe Damas[1], and Pierre Dupont[1,2]

[1] Department of Computing Science and Engineering (INGI)*
Université catholique de Louvain
Place Sainte Barbe, 2
B-1348 Louvain-la-Neuve - Belgium
{bernard.lambeau, christophe.damas, pierre.dupont}@uclouvain.be
[2] UCL Machine Learning Group
http://www.ucl.ac.be/mlg/

**Abstract.** Standard state-merging DFA induction algorithms, such as RPNI or Blue-Fringe, aim at inferring a regular language from positive and negative strings. In particular, the negative information prevents merging incompatible states: merging those states would lead to produce an inconsistent DFA. Whenever available, domain knowledge can also be used to extend the set of incompatible states. We introduce here mandatory merge constraints, which form the logical counterpart to the usual incompatibility constraints. We show how state-merging algorithms can benefit from these new constraints. Experiments following the Abbadingo contest protocol illustrate the interest of using mandatory merge constraints. As a side effect, this paper also points out an interesting property of state-merging techniques: they can be extended to take any pair of DFAs as inputs rather than simple strings.

## 1 Introduction

Deterministic Finite Automaton (DFA) induction is a popular technique to infer a regular language from positive and negative strings defined over a finite alphabet $\Sigma$. Several state-merging algorithms have been proposed to tackle this task, including RPNI [1], EDSM and Blue-Fringe (also known as redBlue) [2]. These algorithms start from a tree-shaped automaton, the so-called prefix tree acceptor (PTA), that accepts the positive sample $S_+$ only, and successively merge states to generalize the induced language. The order in which pairs of states are considered for merging is the key difference between the respective algorithms. In all cases, the generalization is controlled by the negative sample $S_-$ to prevent merging incompatible states. Merging those states would lead to build an inconsistent machine, that is a DFA which accepts at least one negative string.

The availability of negative information is theoretically motivated since positive and negative samples are required to identify in the limit any super-finite class of languages, including the regular language class [3]. The convergence proof of RPNI, for instance,

* This work is partially supported by the Regional Government of Wallonia (Gisele project, RW Conv. 616425).

is related to the definition of a characteristic sample. When RPNI receives as input such a sample, it is guaranteed to output the target machine, that is, the canonical automaton of the target language. Such a characteristic sample includes $\mathcal{O}(n^2)$ negative strings, where $n$ denotes the number of states of the target machine.

When few negative strings are available, one can rely on another kind of knowledge, typically provided by the application domain. For example, when some additional information is available about incompatibilities between states of the initial automaton, state merging algorithms can easily be extended to avoid merging states that are known to be incompatible [4,5]. This technique, sometimes referred to as *state coloring*, may be seen as incompatibility constraints on the induction algorithm.

Our previous work applies grammatical inference to the Requirements Engineering (RE) domain [6]. In such an applicative context, domain-specific information, represented as incompatibility constraints, can be used in conjunction with negative strings to avoid poor generalizations while reducing the induction search space. In the same work, we introduced the QSM algorithm, an active learning extension to RPNI or Blue-Fringe with membership queries [7]. These queries are generated during the induction process and provide additional positive and negative strings, overcoming the limitations of an initially sparse learning sample.

The present work introduces a new kind of constraints, called *mandatory merge constraints*. They form the logical counterpart to the incompatibility constraints. This paper investigates this kind of constraints and how to extend state-merging algorithms to handle them. The extended algorithm, called MSM, has been evaluated using an experimental protocol inspired by the Abbadingo competition [2]. We show experimentally that MSM converges faster than existing algorithms. This result is actually quite straightforward since MSM uses a richer information as input. However, two additional observations are arguably the actual contributions of this work.

Firstly, DFA induction algorithms using state-merging typically include a recursive merging process to reduce the non-determinism of temporary solutions. Such a *merging for determinization* process is often implemented assuming a *tree invariant property*. This property states that, when considering two states to be merged, at least one of them is the root of a tree. Such a property, which holds for RPNI and Blue-Fringe, but interestingly not for EDSM, is a sufficient condition for the determinization process to be finite. We argue here that, even though it is convenient, the tree invariant property is not required as the determinization process stops by itself. The important consequence of this observation is that the initial automaton to be considered no longer needs to be a tree representing a finite sample. Hence MSM naturally gives rise to the *Automaton State Merging* (ASM) algorithm, that is a state-merging induction algorithm which takes as input a positive regular language represented by a DFA $A_+$ and a negative sample $S_-$, and outputs a regular language $L$ with $L(A_+) \subseteq L \subseteq \Sigma^* \setminus S_-$. Secondly, we discuss a natural extension to ASM which takes as input a positive DFA $A_+$ and a "negative" DFA $A_-$ and returns a regular language $L$ with $L(A_+) \subseteq L \subseteq \Sigma^* \setminus L(A_-)$. The initialization of this extended algorithm would immediately detect an inconsistency whenever $L(A_+) \cap L(A_-) \neq \emptyset$.

The rest of this paper is structured as follows. Section 2 briefly reviews the Requirement Engineering application context which originally motivated the present work.

Section 3 reviews the DFA state-merging induction algorithms and the use of incompatibility constraints. Section 4 introduces mandatory merge constraints and describes the MSM algorithm. Section 5 presents the experimental protocol and the results obtained using MSM on synthetic data and on a RE case study. The ASM algorithm to deal with positive and negative automata as inputs is discussed in section 6. Our conclusions and future works are presented in section 7.

## 2 The Synthesis of Software Behavior Models Seen as a DFA Induction Problem

It has been claimed that the hardest part in building a software system is deciding precisely what the system should do. This is the general objective of Requirements Engineering (RE). Automating parts of this process can be addressed by learning behavior models from scenarios of interactions between the software-to-be and its environment. Indeed, scenarios can be represented as strings over an alphabet of possible events and they can be generalized to form a language of acceptable behaviors. Whenever behaviors are modeled by finite-state machines, the problem becomes equivalent to automaton induction from positive and negative strings.

Scenarios are typical examples of system usage provided by an end-user involved in the requirements elicitation process. A simple message sequence chart (MSC) formalism is used for representing scenarios. A MSC is composed of vertical lines representing time-lines associated with agent instances, and horizontal arrows representing interactions among such agents. Figure 1 depicts some scenarios for a simple train system. The system is composed of four agents: a train controller, train motor and doors, and a passenger. For example, the scenario “Start/Stop” expresses that the train controller can start the train and then stop the train from the initial state.

Related works on RE are described in [7]. The technique proposed in the present paper allows one to inductively synthesize behavior models from positive and negative scenarios while taking into account their flowcharting in a high-level Message Sequence Chart (hMSC). A hMSC is a directed graph where each node references a scenario. Edges indicate the acceptable flowcharting of these scenarios, allowing the end-user to reuse scenarios within a specification and to introduce sequences, loops, and alternatives. Figure 1 presents an example of scenario flowcharting for the train example. An induction approach for behavior synthesis is relevant because a $PTA$ can easily be de-

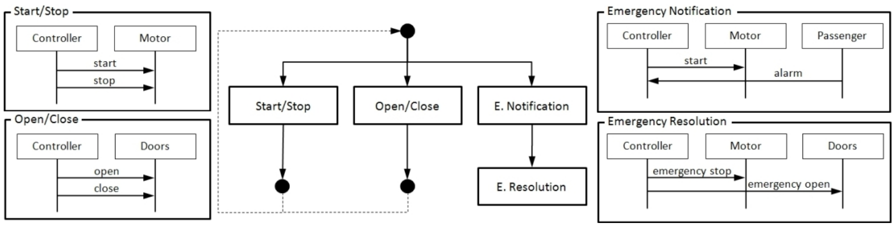


**Fig. 1.** Some scenarios for a simple train system and their flowcharting in a hMSC (center)

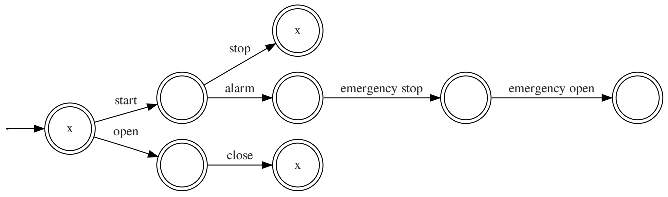


**Fig. 2.** A labeled Prefix Tree Acceptor generated for the train system

rived from the positive and negative scenarios: the transitions drawn as solid lines form a spanning tree of the hMSC resulting in the initial PTA of Figure 2. Moreover, the hMSC provides additional information that must be used during the induction process: dashed lines allow one to identify equivalent system states. A label is associated to those equivalent states in the $PTA$ ($x$ in the example). The MSM algorithm introduced in section 4 ensures that they will be merged by the induction process.

## 3 State-Merging DFA Induction with Incompatibility Constraints

This section briefly reviews the DFA induction problem using state-merging algorithms and incompatibility constraints. The concept of quotient automaton, strongly related with state-merging, is introduced in section 3.1. Section 3.2 presents a family of state-merging algorithms, RPNI being one representative example. Section 3.3 describes one efficient way to handle incompatibility constraints in such algorithms.

### 3.1 State-Merging and Quotient Automaton

State-merging DFA induction algorithms, such as RPNI, start from an initial automaton called a prefix tree acceptor $PTA(S_+)$. It is the largest trimmed DFA accepting exactly $S_+$ (see the automaton on the left of Fig. 3). The generalization operation obtained by merging states is defined through the concept of *quotient automaton*, relative to a partition $\pi$ of the state set of the original atomaton. States belonging to the same subset, or block, of $\pi$ are merged in the quotient automaton (see the automaton on the right of Fig. 3). Any accepting path in the $PTA$ is also an accepting path in $PTA/\pi$. As a quotient automaton corresponds to a particular partition, the set of possible generalizations which can be obtained by merging states of an automaton $A$ can be searched through a lattice of partitions $Lat(A)$ [8].

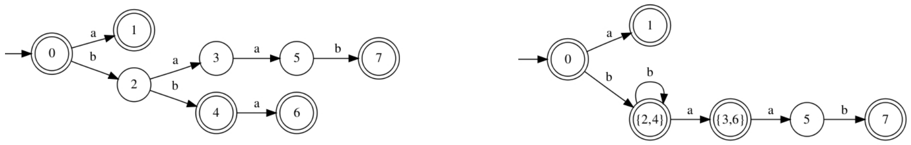


**Fig. 3. (left)** $PTA(S+)$ with $S_+ = \{\lambda, a, bb, bba, baab\}$ and $\lambda$ denoting the empty string; **(right)** a quotient automaton $A = PTA/\pi$ where $\pi = \{\{0\}, \{1\}, \{2,4\}, \{3,6\}, \{5\}, \{7\}\}$. Accepting states are represented as doubly circled nodes.

## 3.2 State-Merging DFA Induction Algorithms

RPNI is a very well known DFA induction algorithm [1]. It can be seen as a particular case of the state-merging algorithm described in Algorithm 1. It takes a positive and a negative sample as input. The first step constructs the PTA[1]. An initial partition is initialized and successively updated by the main loop of the algorithm. At each step of this loop, two blocks of the partition are selected as candidate for merging. These partition blocks precisely define the states of the quotient automaton $PTA/\pi$. In other words, each step can be interpreted as merging two states of a quotient automaton, which forms an intermediate solution of the algorithm. When compatible with the negative sample, this intermediate solution is kept for the next step, otherwise it is simply discarded. The algorithm continues by selecting other blocks to merge, until no more state pairs can be considered.

**Algorithm 1.** A state-merging DFA induction algorithm

```
Algorithm State-Merging DFA Induction Algorithm
Input: A positive and negative sample $(S_+, S_-)$
Output: A DFA $A$ consistent with $(S_+, S_-)$

// Compute a $PTA$, let $N$ denote the number of its states
$PTA \leftarrow$ Initialize($S_+$, $S_-$); $\pi \leftarrow \{\{0\}, \{1\}, \ldots, \{N-1\}\}$

// Main state-merging loop
while $(B_i, B_j) \leftarrow$ ChoosePair($\pi$) do
    $\pi_{new} \leftarrow$ Merge($\pi, B_i, B_j$)
    if Compatible($PTA/\pi_{new}, S_-$) then
        $\pi \leftarrow \pi_{new}$

return $PTA/\pi$

// This function merges two blocks and removes non-determinism recursively
Merge($\pi, B_i, B_j$) begin
    $\pi \leftarrow \pi \backslash \{B_i, B_j\} \cup \{B_i \cup B_j\}$
    while $(B_k, B_l) \leftarrow$ FindNonDeterminism($\pi, B_i, B_j$) do
        $\pi \leftarrow$ Merge($\pi, B_k, B_l$)
    return $\pi$
end
```

The pseudo code of the `Merge` function is shown below the main loop. The first line simply updates the partition $\pi$ by effectively merging the two block arguments. Merging two blocks $B_i$ and $B_j$ may lead to a non-deterministic quotient automaton; the partition is then recursively updated in order to reduce the non-determinism.

The `ChoosePair` function determines which pairs of blocks to consider for merging. In the particular case of the RPNI algorithm, it relies on the standard lexicographical order on strings. The Blue-Fringe algorithm uses an heuristic approach according to the order in which state pairs are chosen, while identifying as soon as possible the states which are incompatible with all their predecessors [2]. For both algorithms, the block $B_i$ in the main loop and, by extension, the block $B_k$ in the `Merge` function, are always the roots of a tree. This is the *tree invariant property* already mentioned in section 1. This property also helps implementing particularly simple and fast algorithms [2].

[1] The reason why the $PTA$ is built using $S_+$ as well as $S_-$ results from a variant motivated in section 3.3.

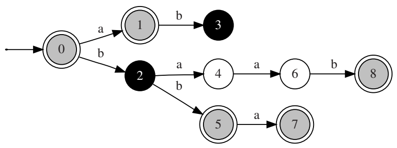


**Fig. 4.** Augmented $PTA$ from the sample $(S_+, S_-) = (\{\lambda, a, bb, bba, baab\}, \{ab, b\})$. The positive sample can be represented by the $PTA$ illustrated on the left of Fig.3. This $PTA$ can be augmented by the negative sample, by coloring states reached by a negative string as black. Accepting states (of positive strings) are filled in grey in this figure.

### 3.3 Handling Incompatibility Constraints

Negative strings play a crucial role in automaton induction algorithms, as they are used to avoid merging incompatible $PTA$ states. When few negative strings are provided in the initial sample $S_-$, alternative sources of knowledge can be used to play a similar role. In particular, knowledge about incompatibilities between states of the $PTA$ can easily be incorporated in the induction process as coloring constraints. As the negative sample $S_-$ can itself be encoded as such constraints, we used this particular example to illustrate the technique in Fig. 4.

To ensure that the solution returned by the induction algorithm correctly rejects $S_-$, it suffices to avoid merging black and grey states of the $PTA$. In other words black and grey states form incompatible pairs. This black/grey coloring can be captured by a partial function $f_{col}(Q) \rightarrow \{grey, black\}$, where $Q$ is the set of $PTA$ states. A quotient automaton $PTA/\pi$ respects the coloring constraint if, $\forall q_1, q_2 \in Q$ such that $f_{col}(q_1) \neq f_{col}(q_2)$ and both are defined, if $q_1 \in B_i$ and $q_2 \in B_j$ then $B_i \neq B_j$. In other words, each block $B$ may contain any number of uncolored states (4 and 6 in our example) but grey and black states must be kept separate from each other. Checking the constraints can be efficiently performed in the `Merge` function. The `Compatible` function in the main loop can thus be replaced by a compatibility test between merged blocks in the `Merge` function. This algorithm is able to detect inadequate solutions during the determinization procedure itself, speeding up the induction process. This improvement is illustrated in section 4 where the MSM algorithm is introduced. More details about this state coloring technique can be found in [5] and examples of RE domain-specific information in [7]. Domain knowledge represented as incompatibility constraints has also been used for subsequential transducer learning [9].

## 4 State-Merging with Mandatory Merge Constraints

Section 2 explains why *mandatory merge constraints* arise in our RE application context. We present here, the MSM (Mandatory State Merging) algorithm, a straightforward adaptation of Algorithm 1 to deal with such constraints.

Mandatory merge constraints are the logical counterpart to the incompatibility constraints. It is interesting to elaborate briefly on the logical differences between them. Consider for example the $PTA$ of Fig. 4. Grey and black states are known to be incompatible: *merging them would lead to a solution that accepts at least one negative string*.

In addition, we assume that some domain-specific knowledge allows the user to state that $q_2$ and $q_6$ in this $PTA$ do in fact represent the same state in the target machine. In our application domain, this kind of knowledge takes the following form: "From the initial state, the software accepts exactly the same future behaviors after the occurrence of event $b$ (state 2) and the sequence of events $baa$ (state 6)". By virtue of the mapping between states of a canonical DFA and residual languages (the "future behaviors" in our example), these states must be merged by the induction process: *not merging them will lead to a solution that does not respect this positive domain knowledge*.

Capturing mandatory merge constraints uses a similar mechanism to *state coloring*, this one being called *state labeling*. Formally, we introduce a partial labeling function $f_{lab}(Q) \rightarrow \{r, s, t, ...\}$, where $Q$ is the set of $PTA$ states and $r$, $s$, $t$ are labels. States with the same label must be merged by the induction process. In our example, the fact that states $q_2$ and $q_6$ must be merged is captured by $f_{lab}(q_2) = f_{lab}(q_6)$. A quotient automaton $PTA/\pi$ respects the labeling constraints if, $\forall q_1, q_2 \in Q$ such that $f_{lab}(q_1) = f_{lab}(q_2)$ and both are defined, if $q_1 \in B_i$ and $q_2 \in B_j$ then $B_i = B_j$.

Given a particular partial coloring function $f_{col}$ and a particular partial labeling function $f_{lab}$, their influence on the induction process is the following. States with different colors *may not* be merged, while states with the same color *may* be merged. States with different labels *may* be merged, while states with the same label *must* be merged. It is worth stressing that the labeling information does not replace the negative knowledge, which is included in the coloring constraints. In the extreme case where all states are correctly labeled but no incompatibility constraints (hence no negative examples) are present, all states can, and actually will, be merged. In short, *labeling* and *coloring* information help together to achieve a good generalization accuracy.

The apparent symmetry between the two types of constraints is however not reflected in the merging process. Coloring (or incompatibility) constraints must be enforced at each step of the algorithm while the labeling (or mandatory) constraints need not be. Indeed, if any intermediate solution violates the coloring constraint, it will also be violated by any quotient automaton of this intermediate solution. In other words, when such a constraint is violated no more path exists in the search space to an adequate solution. This is the reason why intermediate solutions are checked at each step of the algorithm and discarded if required.

In contrast, an intermediate solution not respecting a labeling constraint does not imply that a quotient automaton of this solution would not. A trivial example of this fact is the $PTA$ itself: in our example, states $q_2$ and $q_6$ must be merged while not being originally in the same block, thus initially violating the labeling constraint. Under the hypothesis that labeling and coloring constraints are consistent with each other, some quotient automaton of the $PTA$ may however be consistent with all constraints. Dynamically checking whether an intermediate solution $PTA/\pi$ not satisfying a labeling (or mandatory merge) constraint, can indeed lead to a consistent automaton looks to be a difficult task. The MSM algorithm described below adopts a straightforward approach to satisfy labeling constraints without the need for such a dynamic check.

MSM (see Algorithm 2) is a state-merging induction algorithm ensuring that both kinds of constraints are satisfied. As motivated in section 3.3, incompatibility constraints between states provided by the state coloring (including those provided by the negative sample) are checked in the `Merge` function instead of the main loop; an exception mechanism can be used to handle incompatibility notifications. Secondly, all blocks that must be merged according to the labeling information are merged immediately, that is, before entering the main loop. The `FindSameBlocks` function identifies pairs of blocks in $\pi$ having the same label. These blocks are merged using the `Merge` function. This ensures that potential non-determinism is actually reduced. If *coloring* and *labeling* constraints are inconsistent, the $avoid$ exception is not caught by the algorithm and typically raised to the user. If successful, the first while loop produces a quotient automaton $PTA/\pi$ respecting both coloring and labeling constraints. The partition $\pi$ is then further updated by the usual state-merging loop. While this loop takes care of respecting coloring constraints, the labeling constraints could simply not be violated anymore. Indeed, after the initial phase, states that must be merged are necessarily in the same blocks of $\pi$ and those states will remain merged forever.

**Algorithm 2.** MSM, a DFA induction algorithm that satisfies *incompatibility* (coloring) and *mandatory merge* (labeling) constraints

```
Algorithm MSM
Input: A non-empty initial positive and negative sample $(S_+, S_-)$
Input: Labeling and coloring constraints
Output: A DFA $A$ consistent with $(S_+, S_-)$ and all constraints

// Compute a PTA, let $N$ denote the number of its states
$PTA \leftarrow$ Initialize($S_+$, $S_-$); $\pi \leftarrow \{\{0\},\{1\},...,\{N-1\}\}$

// Merge all states according to labeling constraints
while $(B_i, B_j) \leftarrow$ FindSameBlocks($\pi$) do
    $\pi \leftarrow$ Merge($\pi, B_k, B_l$)

// Main state-merging loop
while $(B_i, B_j) \leftarrow$ ChoosePair($\pi$) do
    try
        $\pi \leftarrow$ Merge($\pi, B_i, B_j$)
    catch avoid
        // next state pair to consider

return $PTA/\pi$

// This function merges two blocks and removes non-determinism recursively
// while checking coloring constraints
Merge($\pi, B_i, B_j$) begin
    if Incompatible($B_i, B_j$) then
        raise avoid
    $\pi \leftarrow \pi\backslash\{B_i, B_j\} \cup \{B_i \cup B_j\}$
    while $(B_k, B_l) \leftarrow$ FindNonDeterminism($\pi, B_i, B_j$) do
        $\pi \leftarrow$ Merge($\pi, B_k, B_l$)
    return $\pi$
end
```

The implementation of MSM looks *a priori* straightforward. Starting from the $PTA$, a DFA *A* is built by merging all states that must be merged. Next, the main state-merging loop is executed from *A*. It is however worth stressing that the *tree invariant property* does not hold in MSM. This observation may require to significantly review the actual implementation of this algorithm. Interestingly, the main merging loop and the `Merge`

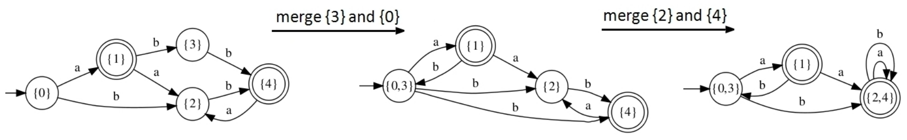


**Fig. 5.** Recursive determinization process. States {3} and {0} of an arbitrary DFA are merged, which causes a non-determinism on letter *b* from state {0,3}. The destination states {2} and {4} are subsequently merged to reduce the non-determinism.

function can be implemented without the tree invariant property because the recursive determinization process stops naturally on the first DFA encountered. This observation allows one to start from an arbitrary DFA and, as soon as non-determinism occurs, to reduce it. Figure 5 gives an example of such a recursive operation.

## 5 Evaluation

MSM has been evaluated on synthetic data as well as on the RE train case study briefly introduced in section 2. Sections 5.1 and 5.2 discuss the respective results of these experiments.

### 5.1 Experiments on Synthetic Data

To evaluate MSM on synthetic data, we used an experimentation protocol inspired from the Abbadingo contest [2]. In our current implementation, MSM is equivalent to RPNI when no mandatory merged constraints are present. In other words, the merging order is exactly the one of RPNI if no labeling constraints are considered. In this context, the objective of this evaluation is mostly to quantify the proportion of domain-specific information required to get better generalization results. The experimentation protocol that we used to achieve this objective is outlined below.

Experiments are made on randomly generated target DFAs with 32 and 64 states and an alphabet of 2 letters. Accepting states are chosen randomly by flipping a fair coin. These automata are trimmed to remove unreachable states and minimized to obtain canonical target machines. The number of states of a DFA generated using this procedure is approximately 3/5 of the requested size, which has been increased accordingly. As in Abbadingo, if the depth of the resulting automaton is not equal to $p = 2 * log_2(n) - 2$, it is simply discarded.

A learning and testing set for a target DFA with $n$ states consists of $n^2$ randomly generated strings. These strings are generated using a uniform distribution over the collection of all binary strings of length $[0, p + 5]$. This set is randomly divided into two samples of the same size: a learning sample on which MSM is run and a testing sample used to measure the adequacy of the resulting solution. Strings of the learning sampled are labeled as positive or negative according to the target DFA. MSM is run on increasing proportions of the learning sample.

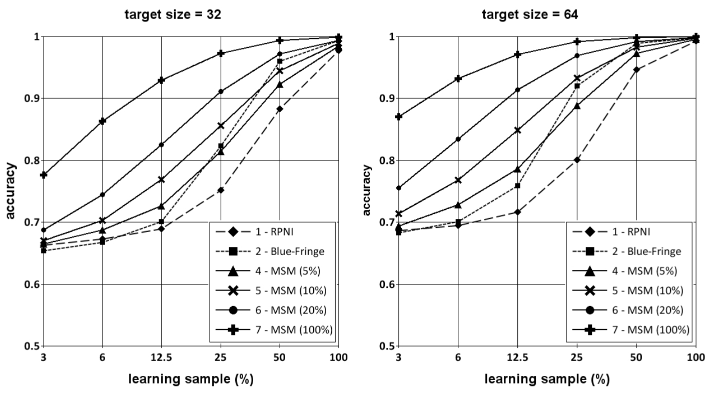


**Fig. 6.** Classification accuracy for RPNI, Blue-Fringe and MSM

In order to simulate domain-specific information leading to mandatory merge constraints, unique labels are associated to randomly chosen states of the target DFA. Increasing proportions of the number of states labeled in this way have been used: 5%, 10%, 20% and 100%. States of the $PTA$ used by MSM are labeled by jointly visiting it with the target DFA and reporting encountered labels. This labeling constrains the induction process as explained in section 4.

Figure 6 reports the proportion of independent test samples correctly classified while increasing the learning sample. Curves in this plots correspond to executions of RPNI, Blue-Fringe and MSM with different labeling proportions. Each point in these plots is the average value computed over 200 independent runs. MSM overcomes RPNI on all executions, which was actually expected, but illustrates experimentally that forcing to merge initially some (correctly labeled!) states does not prevent from converging. 5% of labeling information is comparable, from the point of view of the generalization accuracy, to the use of the Blue-Fringe heuristic for selecting state pairs to be merged. Beyond this proportion, the accuracy continues to increase. Interestingly, it is already visible when the sample is sparse. This fact is particularly helpful in our RE context, where learning sample is initially provided by an end-user. Moreover, it is worth noting that, as pointed out in section 4, the identification of the target does not reduce to a trivial problem even with 100% of labeling information when only few negative examples are available.

Although not yet implemented, we are confident that using mandatory merge constraints would also improve the generalization accuracy of the Blue-Fringe and QSM algorithms (the interested reader may refer to [7] for an experimental comparison between RPNI, Blue-Fringe and QSM without labeling constraints).

## 5.2 Experiments on a RE Case Study

An accuracy gain is also expected when using MSM for behavior model synthesis. In order to quantify this gain, the algorithm has been evaluated on an extended version of the train system introduced in section 2. In this respect, the evaluation protocol is slightly different from the one presented in the previous section: the target model of the train system has been built manually as a DFA of 18 states with an alphabet of size 10 (see Figure 7). A typical collection of scenarios has also been built for the system and represented as an augmented PTA: 9 positive and 5 negative strings for a total of 55 states.

Mandatory merge constraints are defined by labeling pairs of equivalent PTA states. These pairs are chosen following a "loop identification" heuristic, representative of the way such a specification is incrementally built by an end-user using an hMSC. For instance, opening then closing the doors from the initial state (without any intermediate event) naturally returns in the initial state (see the loop between states 0 and 2 in Figure 7 and the way this loop is easily represented in the hMSC of Figure 1). However, some loops are less natural to identify from the scenarios: the sequence of events ($leaving$, $high$, $approaching$, $low$, $atstation$) form a loop starting from state 3 for example. Equivalent state pairs of the PTA identified in this way have been classified in four categories, following the expected difficulty for an end-user of discovering them in the scenarios. MSM has been evaluated on increasing proportion of mandatory merge constraints, following this classification.

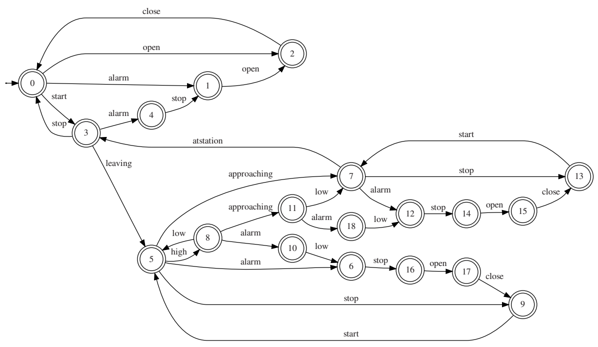


**Fig. 7.** Target model of the train system

**Table 1.** Classification accuracy obtained with different setups on the train case study. The number of labeling constraints $|lab|$ refers to the number of PTA states pairs declared to be equivalent.

| Algorithm | $\|lab\|$ | Accuracy |
|---|---|---|
| RPNI | - | 0.55 |
| BlueFringe | - | 0.83 |
| MSM | 0 | 0.55 |
| | 3 | 0.71 |
| | 6 | 0.73 |
| | 10 | 0.88 |
| | 15 | 0.90 |

Table 1 compares the accuracy of the DFA induced using RPNI and BlueFringe as well as MSM with an increasing number of mandatory merge constraints. The reported accuracy is the average classification rate computed over 10 independent test samples, each one containing 80 (positive or negative) strings. The results confirm what has been observed on synthetic data. Increasing the number of mandatory merge constraints (that is, enriching the hMSC with additional transitions) leads to a better accuracy, outperforming BlueFringe when such information is rich enough. The results also show that no algorithm has been able to identify perfectly the target DFA on such sparse samples. It is worth noting that, for the need of this evaluation, only few sources of negative information have been used while such sources do exist in this application domain [7]. Further improvements could also be obtained by using mandatory merge constraints with the BlueFringe search order.

## 6 DFA Induction from Positive and Negative DFAs (Deterministic Finite Automata) as Inputs

Relaxing the tree invariant property has additional benefits which we discuss in this section. The main state-merging loop of MSM (see Algorithm 2) actually generalizes a language represented by an arbitrary DFA, under the control of all negative knowledge represented as incompatibility constraints. It is possible to factor out the state-merging loop from MSM as a new algorithm ASM (for Automaton State Merging), which generalizes a positive DFA $A_+$ taken as input, under the control of a negative sample $S_-$, ignoring here other incompatibility constraints for the simplicity of the discussion. The pseudo-code of ASM is given in Algorithm 3.

**Algorithm 3.** ASM, a DFA induction algorithm that generalizes a positive DFA $A_+$ under the control of a negative sample $S_-$

```
Algorithm ASM
Input: A positive DFA $A_+$ and a negative sample $S_-$
Output: A DFA $A$ consistent with $(A_+, S_-)$

// Augment the automaton $A_+$ with states
// marked/added from $S_-$
$M \leftarrow$ Augment($A_+$, $S_-$)

// Compute the natural order on $M$
$\pi \leftarrow$ NatOrder($M$)

// Main state-merging loop
$\pi \leftarrow$ Generalize($\pi$)

return $M/\pi$
```

The first step of this algorithm augments $A_+$ with the negative sample $S_-$ in a way similar to the augmentation of a $PTA(S_+)$. Each negative string can be decomposed as $uv$, where $u$ is the longest prefix already present in $A_+$ and reaches a state $q$, and $v$ is the suffix of this negative string. When $v$ is empty, $q$ is marked as a negatively accepting (= black) state if not yet marked as positively accepting; if $q$ is already a positively accepting (= grey) state, an inconsistency error between $A_+$ and $S_-$ is reported to the user. When $v$ is not empty, a new branch rooted at $q$ is added to the automaton, ending in a new negatively accepting state.

The function `NatOrder` computes the natural order of the states of $M$ using a breadth first search of the states and numbering each of them when encountered. The search ends when each state has been reached. We assume here for simplicity that this function returns the trivial partition $\pi$ with each state in its own block, the blocks being naturally ordered. The `Generalize` function corresponds to the main state-merging loop and returns the updated partition.

ASM is the actual algorithm we use for the synthesis of behavior models in our RE application domain. Unfolding the hMSC as a labeled $PTA$ is in fact not required and one can directly generalize, using ASM, the automaton that would be produced from the hMSC by the deductive technique of [10].

The ASM algorithm itself may be further extended. Indeed, the extension which consists in replacing finite positive string $S_+$ by a regular language represented as a DFA $A_+$, can also be applied to the negative sample. This lead to an induction algorithm ASM$^*$, which takes as input both a positive DFA $A_+$ and a negative DFA $A_-$. In ASM$^*$, the `Augment` function produces the composition of $A_+$ and $A_-$ as a colored automaton $M$. More precisely, the state space of $M$ is the product of the states of $A_+$ and $A_-$ (extended whenever necessary to be complete DFAs). A state of $M$ is positively accepting if the corresponding state is accepting in $A_+$. Similarly, it is negatively accepting if the corresponding state in $A_-$ is accepting. If both conditions hold at the same time for at least one state of $M$, an inconsistency error between $A_+$ and $A_-$ is reported to the user. Figure 8 illustrates this composition mechanism on a simple example. The natural order is computed on $M$. The main state-merging loop is then executed while checking for the coloring constraints as usual in the recursive `Merge` function.

There could be no need to compute explicitly the product automaton $M$ since the associated coloring constraints, which follow from the common prefixes between $A_+$ and $A_-$, can be computed and updated dynamically, as nicely shown in [5]. We believe however that it is useful to consider such a product automaton for defining the state ordering relation computed in `NatOrder`. Doing so, the prefixes of the negative strings in $L(A_-)$, including possibly some prefixes that do not belong to $L(A_-)$ themselves, can be used to define the search order. In this sense, strict prefixes of positive and negative strings are considered in the same way. This is indeed the natural extension to the augmented $PTA(S_+, S_-)$ used to define the search order in the Blue-Fringe algorithm. We also note that [5] generalizes those ideas to non-deterministic automata. However the authors do not stress the possibility to start the whole induction process from both a positive and a negative automaton, as a consequence of relaxing the tree invariant property.

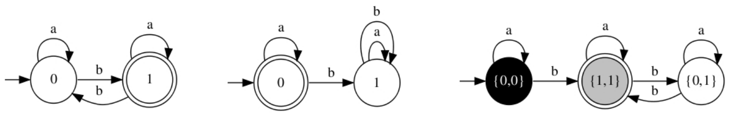


**Fig. 8.** Composition of $A_+$ (left) and $A_-$ (middle) to get a product automaton $M$ (right) with coloring constraints

Finally, it is worth noting that RPNI2, the incremental version of the RPNI algorithm introduced in [11], unfolds the current DFA when a new negative example is received and wrongly accepted by the current machine. The unfolded DFA is subsequently merged following a different path in the search space. As such, RPNI2 is already able to restart the generalization process from a DFA not restricted to be a tree. However the tree invariant property is satisfied in RPNI2 since the unfolding is guaranteed to go back to a temporary solution that RPNI would have considered if run on the updated samples.

## 7 Conclusion and Future Work

Coloring constraints are classically used in automaton induction techniques to control the state merging process while generalizing the positive sample. Such coloring constraints define which states are incompatible, that is, cannot be merged without giving rise to an inconsistent machine. We introduce here mandatory merge constraints, implemented using a partial labeling function, as the logical counterpart to the incompatibility constraints. We propose the MSM algorithm, a natural extension to state-merging algorithms such as RPNI or Blue-Fringe, that can deal both with coloring and labeling constraints. We present experimental comparisons between MSM, RPNI and Blue-Fringe following a protocol inspired by the Abbadingo competition.

The MSM extension looks straightforward from an algorithmic point of view but it actually relaxes the tree invariant property. This property states that, among two states considered for merging, at least one is always the root of a tree-shaped (sub-)automaton. The tree invariant property is often assumed in DFA induction algorithms when recursively merging pairs of states to reduce non-determinism. However such a merging for determinization process naturally stops by itself and the tree invariant property is thus not required. As a consequence, the MSM algorithm gives rise to the ASM algorithm that takes a DFA $A_+$ and a negative sample $S_-$ as inputs. ASM is the actual induction algorithm we use in our Requirements Engineering application domain which motivated, in the first place, the definition of mandatory merge constraints. We also describe the ASM$^*$ algorithm, which further extends ASM, and takes as input both a positive and a negative DFA.

Our future work includes several points. Our current implementation of MSM, and hence of ASM, relies on the RPNI search order. MSM and ASM can also be adapted to the Blue-Fringe strategy typically with the EDSM scoring function to adapt the search order. Doing so would only require to compute the scoring function between state pairs as a side product of the recursive merging operation applied here on general graphs. In this regard, the implementation would be similar to the original EDSM algorithm (*i.e.* without the Blue-Fringe strategy) but with coloring and labeling constraints. A further step is to extend the QSM algorithm [7] to add the active learning feature.

ASM$^*$ raises interesting theoretical questions since inferring from a positive and a negative DFA no longer fits exactly in the identification in the limit framework. The definition of a characteristic sample would need to be adapted as well as the experimental protocol.

## References

1. Oncina, J., García, P.: Identifying regular languages in polynomial time. In: Bunke, H. (ed.) Advances in Structural and Syntactic Pattern Recognition. Series in Machine Perception and Artificial Intelligence, vol. 5, pp. 99–108. World Scientific, Singapore (1992)
2. Lang, K., Pearlmutter, B., Price, R.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 1–12. Springer, Heidelberg (1998)
3. Gold, E.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
4. Coste, F., Nicolas, J.: How considering incompatible state mergings may reduce the DFA induction search tree. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 199–210. Springer, Heidelberg (1998)
5. Coste, F., Fredouille, D., Kermorvant, C., de la Higuera, C.: Introducing domain and typing bias in automata inference. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 115–126. Springer, Heidelberg (2004)

6. Damas, C., Lambeau, B., Dupont, P., van Lamsweerde, A.: Generating annotated behavior models from end-user scenarios. IEEE Transactions on Software Engineering 31(12), 1056–1073 (2005)
7. Dupont, P., Lambeau, B., Damas, C., van Lamsweerde, A.: The QSM algorithm and its application to software behavior model induction. Applied Artificial Intelligence 22, 77–115 (2008)
8. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 25–37. Springer, Heidelberg (1994)
9. Oncina, J., Varó, M.A.: Using domain information during the learning of a subsequential transducer. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 301–312. Springer, Heidelberg (1996)
10. Uchitel, S., Kramer, J., Magee, J.: Synthesis of behavorial models from scenarios. IEEE Transactions on Software Engineering 29(2), 99–115 (2003)
11. Dupont, P.: Incremental regular inference. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 222–237. Springer, Heidelberg (1996)

# Using Multiplicity Automata to Identify Transducer Relations from Membership and Equivalence Queries

Jose Oncina

Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, Spain
oncina@dlsi.ua.es

**Abstract.** Multiplicity Automata are devices that implement functions from a string space to a *field*. Usually the real number's field is used. From a learning point of view there exists some algorithms that are able to identify any multiplicity automaton from membership and equivalence queries.

In this work we realize that those algorithms can also be used if the algebraic structure of a field is relaxed to a *divisive ring* structure, that is, the commutativity of the product operation is dropped.

Moreover, we define an algebraic structure, which is an extension of the string monoid, that allows the identification of any transduction that can be realized by finite state machines without empty-transitions.

## 1 Introduction

In the same way a language is defined as a subset (usually infinite) of strings, a transducer can be defined as a subset of pairs or strings. The first string is interpreted as the *input* string and the second as the *output* string, *i.e.* in a translation task the pair (*"to be"*, *"ser"*) can represent that the English verb *"to be"* can be translate to Spanish by the verb *"ser"*.

One of the biggest classes of transductions that are known to be identifiable are the *subsequential functions*. Those functions can be describe as the set of the transductions that can be implemented by deterministic finite state machines in which the arcs and the states are labeled with strings of the output string space. The translation of a string is the concatenation of the strings in the labels of the arcs and the final state used in the parsing of the string (note that since the automaton is deterministic there is at most one path).

In this work we are interested in learning transducers in the exact learning model. In this model, proposed by Angluin in 1988 [Ang88], the learner is allowed to actively search information by asking queries to a teacher. Two types of queries are allowed:

– *membership* queries. When the learner can ask for the translation of some sentence.

– *equivalence* query. When the learner thinks it has a suitable hypothesis, it can ask the teacher if it is correct. If the model is correct the teacher answers YES and the process stops. If it is incorrect, the teacher answers with a counter example.

In 1996 Vilar [Vil96] proposed an algorithm to efficiently identify any subsequential transducer in this model.

The most important drawback of the subsequential functions is that they can not cope with ambiguities. For example, the verb *"to be"* can also be translated to Spanish by the verb *"estar"*, but subsequential functions, as they are based on deterministic machines, are unable to give several options.

In this work we are going to describe a class of transductions that includes properly the subsequential functions and permits the expression of ambiguous transductions. This class includes all the transductions that can be performed by non deterministic $\epsilon$-free automata where the edges and the states can have several labels.

The proposed algorithm is a reinterpretation of an algorithm to identify Multiplicity Automata (MA) [BBB+00] in the exact learning framework. Usually MA implement string to real functions. MA are very similar to stochastic automata where there are no restrictions to force the function to be a distribution probability. They can be described as a non deterministic automata with labels in the edges and states (usually real numbers). The value assigned to a string is the sum over all possible parses of the string of the product of the labels in the edges and the final state used in each parse.

Our idea relays in substituting the field of the real numbers (with its multiplication and addition) that is usually used in MA by an alphabet with the concatenation playing the role of the multiplication and the inclusion in a multiset playing the role of the addition. Note that the commutativity is lost when replacing the product of reals by the concatenation of strings. That means we are not longer working in a field but, with some extensions described later, in a *divisive ring*. It is easy to check that, in the demonstration of the properties of his algorithm, Beimel *et al* did not use the commutativity of the product, and therefore, their results remains valid in the case of *divisive rings*.

## 2 Notation

### 2.1 String Expressions

Let $\Sigma = \{a, b, \dots\}$ be a finite set or *Alphabet*. The set of *string expressions* over $\Sigma$ ($E(\Sigma)$) is defined as follows:

– $\epsilon \in E(\Sigma)$ and $\emptyset \in E(\Sigma)$
– let $a \in \Sigma \Rightarrow a \in E(\Sigma)$
– let $x \in E(\Sigma) \Rightarrow -x \in E(\Sigma)$
– let $x \in E(\Sigma) \Rightarrow x^{-1} \in E(\Sigma)$
– let $x, y \in E(\Sigma) \Rightarrow x + y \in E(\Sigma)$
– let $x, y \in E(\Sigma) \Rightarrow x \cdot y \in E(\Sigma)$

Let $x, y, z \in E(\Sigma)$, we define the following relation of equivalence:

- $(x \cdot y) \cdot z \equiv x \cdot (y \cdot z)$ product associativity
- $(x + y) + z \equiv x + (y + z)$ addition associativity
- $x + y \equiv y + x$ addition commutativity
- $x \cdot (y + z) \equiv x \cdot y + x \cdot z$ product distributivity
- $x + \emptyset \equiv \emptyset + x \equiv x$ addition neutral element
- $x \cdot \epsilon \equiv \epsilon \cdot x \equiv x$ product neutral element
- $x + (-x) \equiv \emptyset$ addition inverses
- $x \cdot x^{-1} \equiv \epsilon$ product inverses

Let we call $[\Sigma]$ the collection of the equivalence classes in $E(\Sigma)$. Then $([\Sigma], +, \cdot)$, in abstract algebra, has a structure of *divisive ring*, that is, a *field* where the product operator is not commutative.

The product is interpreted as the usual concatenation of strings (this is the reason to avoid the commutativity) where $\epsilon$ (the neutral element) represents the null string. The set has been enriched with concatenation symmetric elements, then it is expected to have strings like $ab^{-1}a$. We can think in this strings as "intermediate" results. It should be assured that, at the end of all the computations, no strings with "special symbols" appears. In the sequel, we use the usual juxtaposition notation ($xy$ instead of $x \cdot y$).

The addition is interpreted as a multiset inclusion. Then the expression $x + y$ represents a multiset with two elements, $x$ and $y$, $x + x + x$ is a multiset with three elements (that are equal), $x + y + -y$ a multiset with just one element (the $x$) since $y - y \equiv \emptyset$ and $y + \emptyset \equiv y$.

## 2.2 Multiplicity Automata

**Definition 1 (Multiplicity Automata).** *Let $\Sigma$ be an alphabet and let $K$ be a ring. A* Multiplicity Automaton *(MA) is a 3-tuple $(\lambda, \mu, \gamma)$ such that:*

- *$\lambda \in K^{1\times n}$ (a $1 \times n$ matrix with values in $K$)*
- *$\gamma \in K^{n\times 1}$ matrix (a $n \times 1$ matrix with values in $K$)*
- *$\mu$ is morphism of monoids $\mu : \Sigma^* \to K^{n\times n}$ (return a $n \times n$ matrix)*

That is, $\mu(\lambda) = I$ (the unit matrix), and $\forall x, y \in \Sigma^*, \mu(xy) = \mu(x)\mu(y)$. Then, $\mu$ can be represented as a set of $|\Sigma|$ $K^{n\times n}$ matrices.

Let $\alpha = (\lambda, \mu, \gamma)$ be a multiplicity automaton, we can define the function performed by $\alpha$ as $f_\alpha : \Sigma^* \to K$ such that:

$$f_\alpha(x) = \lambda\mu(x)\gamma$$

A Multiplicity automata $\alpha = (\lambda, \mu, \gamma)$ can also be interpreted as a weighted non deterministic automata with $n$ states. Where $\lambda_i$ is the weight of beginning in state $q_i$, $[\mu(a)]_{i,j}$ is the weight of the arc that goes from state $q_i$ to state $q_j$ with the symbol $a$ and $\gamma_i$ is the weight of ending in state $q_i$. The weight of a path is computed as the product of the weight of the start state times the weights of the arcs used in its parsing times the weight of the ending state. The weight of a string is the sum of the weights of all the possible paths of the string in the automaton.

It is easy to see that any multiplicity automata function can be implemented by a multiplicity automaton with only one initial state and with at most one more state than the original. Then, without loss of generality, the vector $\lambda$ in the definition can be fixed to a vector such that $\lambda_1 = \epsilon$ and $\lambda_i = \emptyset, 1 < i \leq n$.

In our case instead of a generic ring we are going to use the divisive ring of the string expressions.

*Example 1.* Let $\Sigma_i = \{a\}$ and $\Sigma_o = \{0, 1\}$ be respectively the input and output alphabets, let $f : \Sigma_i^* \to [\Sigma_o]$ be a function such that:

$$f(a^n) = \begin{cases} 0^n & \text{if } n \text{ is odd} \\ 1^n & \text{if } n \text{ is even} \end{cases} \tag{1}$$

It is easy to see that the MA depicted in figure 1 realizes this function.

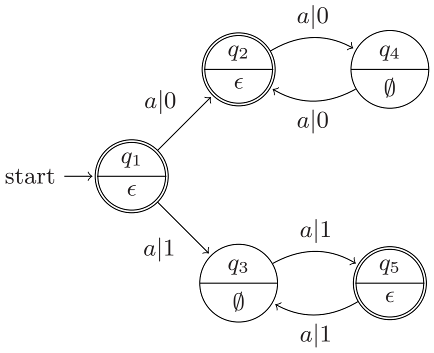


**Fig. 1.** MA for function in equation 3

*Example 2.* The matrix representation of the automaton in figure 1 is:

$$\lambda = \begin{pmatrix} \epsilon & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix} \qquad \mu_a = \begin{pmatrix} \emptyset & 0 & 1 & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & 0 & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & 1 \\ \emptyset & 0 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & 1 & \emptyset & \emptyset \end{pmatrix} \qquad F = \begin{pmatrix} \epsilon \\ \epsilon \\ \emptyset \\ \emptyset \\ \epsilon \end{pmatrix} \tag{2}$$

*Example 3.* It is not very difficult to see that the function in equation 3 can also be realized by the MA automaton in figure 2.

It is easy to see that any transducer based on a non deterministic $\epsilon$-free automaton with a finite number of strings in the edges or states can be represented as a MA over the string expressions divisive ring.

Let we see an example to illustrate that.

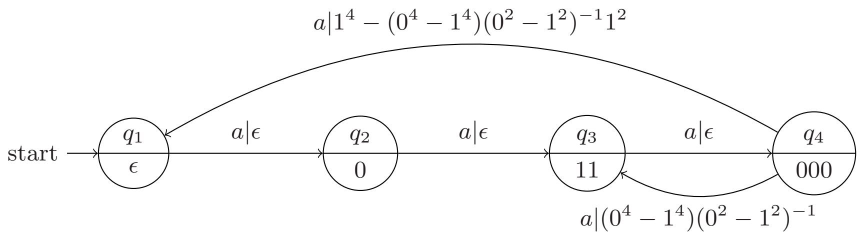


**Fig. 2.** A minimum size automaton that realizes equation 1 function

*Example 4.* Let $\Sigma_i = \{a\}$ and $\Sigma_o = \{0\}$ be respectively the input and output alphabets, let $f : \Sigma_i^* \to [\Sigma_o]$ be a function such that:

$$f(a^n) = \sum_{i=0}^{n} 0^i \tag{3}$$

That is:

$$\begin{aligned} f(\epsilon) &= \epsilon && (\equiv \{\epsilon\}) \\ f(a) &= \epsilon + 0 && (\equiv \{\epsilon, 0\}) \\ f(aa) &= \epsilon + 0 + 00 && (\equiv \{\epsilon, 0, 00\}) \\ &\dots \end{aligned}$$

It is easy to see that the MA depicted in figure 3 realizes this function.

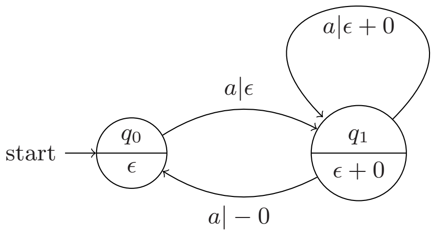


**Fig. 3.** MA for function in equation 3

### 2.3 Hankel Matrix

The MA inference algorithm that we are going to use relies on some properties of the Hankel matrix. Although the following definitions and theorems are stated for fields we can realize the the commutativity of the product is never used and then, they still valid for divisive rings. Part of the following material can be found in some Beimel *et al* papers [BBB+96] [BBB+00], we just transliterate it for divisive rings to make the paper more self content.

Let $\mathcal{D}$ be a divisive ring (A field where the product is not commutative), $\Sigma$ be an alphabet, $\epsilon$ be the empty string, and $f : \Sigma^* \rightarrow \mathcal{D}$ be a function. The *Hankel matrix* of the function $f$ is an infinite matrix $F$ where each of its rows and columns are indexed by strings in $\Sigma^*$. The $(x, y)$ entry of $F$ contains the value $f(xy) \in \mathcal{D}$

We use $F_x$ to denote the $x$th row of $F$. The $(x, y)$ entry of $F$ may be therefore denoted as $F_x(y)$ or as $F_{x,y}$.

*Example 5.* The Hankel matrix of the function in equation 1 is:

$$F = \begin{pmatrix} \epsilon & 0 & 11 & 000 & 1111 & \ldots \\ 0 & 11 & 000 & 1111 & 00000 & \ldots \\ 11 & 000 & 1111 & 00000 & 111111 & \ldots \\ 000 & 1111 & 00000 & 111111 & 0000000 & \ldots \\ 1111 & 00000 & 111111 & 0000000 & 11111111 & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \end{pmatrix} \tag{4}$$

The following theorem of Carlyle and Paz [CP71] and Fliess [Fli74] is a fundamental theorem from the theory of formal series. Although it was stated for field it is easy to check that is is also valid for divisive rings as we state here.

**Theorem 1.** *Let $f : \Sigma^* \rightarrow \mathcal{D}$ such that $f \not\equiv 0$ and let $F$ be the corresponding Hankel matrix. Then, the size $r$ of the smallest MA $\alpha$ such that $f_\alpha \equiv f$ satisfies $r = rank(F)$ (over the divisive ring $\mathcal{D}$)*[1]

The importance of the theorem is double: first, it relates the size of the minimal automaton for $f$ to the rank of its Hankel matrix. And second, the proof is constructive. It gives a way to build a MA from any finite rank Hankel Matrix.

Given a function $f : \Sigma^* \rightarrow \mathcal{D}$ such that the corresponding matrix $F$ has finite rank $r$, let $F_{x_1}, F_{x_2}, \ldots, F_{x_r}$ be $r$ linearly independent rows of $F$ (*i.e.* a basis) corresponding to strings $x_1, x_2, \ldots, x_r$. (since $f \not\equiv 0$, it holds that $F_\epsilon \neq 0$, then $F_\epsilon$ can always be an element of the basis. Then, we take $x_1 = \epsilon$).

Then the MA $\alpha = (\lambda, \mu, \gamma)$ that realizes the function $f_\alpha$ is:

– $\lambda = (\epsilon, \emptyset, \ldots, \emptyset)$.
– $\gamma = (f(x_1), \ldots, f(x_r))^t$.
– for every $a \in \Sigma$, define the $i$th row of the matrix $\mu(a)$ as the (unique) coefficients of the row $F_{x_i}a$ when expressed as a linear combination of $F_{x_1}, \ldots, F_{x_r}$. That is,

$$F_{x_i a} = \sum_{j=1}^{r} [\mu(a)]_{i,j} F_{x_j} \tag{5}$$

*Example 6.* It can be show that $F_\epsilon, F_a, F_{aa}$ and $F_{aaa}$ in the Hankel matrix of equation 4 forms a basis.

We are going to show that $F_{aaaa}$ can be expressed as a linear combination of $F_\epsilon, F_a, F_{aa}$ and $F_{aaa}$.

[1] The demonstration of the theorem can also be found in [BBB+96] [BBB+00].

That is we have to solve the system of equations:

$$\begin{array}{llll} \alpha_1 & +\alpha_2 0 & +\alpha_3 1^2 & +\alpha_4 0^3 = 1^4 \\ \alpha_1 0 & +\alpha_2 1^2 & +\alpha_3 0^3 & +\alpha_4 1^4 = 0^5 \\ \alpha_1 1^2 & +\alpha_2 0^3 & +\alpha_3 1^4 & +\alpha_4 0^5 = 1^6 \\ \alpha_1 0^3 & +\alpha_2 1^4 & +\alpha_3 0^5 & +\alpha_4 1^6 = 0^7 \end{array}$$

It is straight forward to use the Gauss method to solve the system (paying attention to not use the product commutativity) and find the solution:

$$\alpha_4 = \emptyset$$
$$\alpha_2 = \emptyset$$
$$\alpha_3 = (0^4 - 1^4)(0^2 - 1^2)^{-1}$$
$$\alpha_1 = 1^4 - (0^4 - 1^4)(0^2 - 1^2)^{-1}1^2$$

Observe that these are the values that were used to depict the MA in figure 2.

## 3 The Beimel *et al* Algorithm

The algorithm works using a finite version of the Hankel matrix $\hat{F}$. Let $X$ and $Y$ be two sets where the indexes of the finite version of the Hankel matrix are stored.

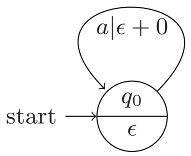
The algorithm works as follows:

```
1. $X = \{x_1 = \epsilon\}, Y = \{y_1 = \epsilon\}$ and $\ell = 1$
2. Build a MA $\alpha = (\lambda, \mu, \gamma)$ using theorem 1
3. Ask an equivalence query.
   If the answer is YES halt with output $\alpha$.
   Otherwise, let $z$ be the counterexample.
   (a) Find (using membership queries) a string $wa$ which is a prefix of $z$ such
       that:
        i. $\hat{F}_w = \sum_{i=1}^{\ell}[\lambda\mu(w)]_i \hat{F}_{x_i a}$; but
       ii. there exists a $y$ such that: $\hat{F}_{wa} \neq \sum_{i=1}^{\ell}[\lambda\mu(w)]_i \hat{F}_{x_i a}(y)$
   (b) $X = X \cup \{x_{\ell+1} = w\}$, $Y = \{y_{\ell+1} = ay\}$, $\ell = \ell + 1$
   GO TO step 2
```

Beimel *et al* showed that the algorithm works in $O((|\Sigma_i| + m)rM(r))$ time using $r$ equivalence queries and $O((|\Sigma_i| + \log m)r^2)$ membership queries. Where $\Sigma_i$ is the input alphabet, $r$ is the rank of the Hankel matrix, $M(r)$ is the complexity of multiplying two $r \times r$ matrices ($O(r^{2.376})$) and $m$ is the length of the longest counter example.

Once more, it can be checked in the work of Beimel *et al* that the commutativity of the product is not used and then, the algorithm still valid in divisive rings.

Note that in our case, the equivalence query should return a string expression describing *all* the possible output strings and the counter example of an equivalence query should return a pair (string, string expression) such that the string expression is a description of all the possible transduction of the input string.



$$\begin{aligned} f(\epsilon) &= \epsilon \\ f(a) &= \epsilon + 0 \\ f(aa) &= (\epsilon + 0)(\epsilon + 0) = \epsilon + 0 + 0 + 00 \\ &\dots \end{aligned}$$

**Fig. 4.** MA that does no realize function in equation 3

*Example 7.* Let we try to find a transducer for the function in equation 3:

In such case we need 2 states and the system to solve is:

$$\begin{array}{lll} \alpha_1 \epsilon & +\alpha_2(\epsilon + 0) & = \epsilon + 0 + 0^2 \\ \alpha_1(\epsilon + 0) & +\alpha_2(\epsilon + 0 + 0^2) & = \epsilon + 0 + 0^2 + 0^3 \end{array}$$

The solution is:

$$\alpha_1 = -0 \qquad\qquad \alpha_2 = \epsilon + 0$$

And the transducer in figure 2 is obtained.

Note that the transducer in figure 4 does not produces the transduction of equation 3 since some of the output strings have a different number of repetitions (multiplicity).

## 4 Conclusions and Open Questions

This work shows how to use an algorithm devised to learn multiplicity automata from membership and equivalence queries to identify transducer relations.

The proposed method can identify the class of transductions than can be expressed as finite state (and arc) machines with no $\epsilon$ transitions.

As this is a first step to deal with ambiguous transductions it remains many problems to solve in order to be able to apply similar technique in more realistic situations:

– The way the membership and equivalence queries should be answered is too demanding. Information about *all* the transductions for the involved input string should be provided. Can we still be able to learn if only information about just one transduction is provided in each query?
– Since the method assures the identification, if the target function does not produces strings with inverse symbols, neither the produced function will do. But it can produce a complex string expression that, when simplified, is just a plain string (as happens in the MA in figure 2). Does it exist a general method to simplify and compare string expressions? Does it exist a method to know if a multiplicity automaton produces only plain strings? If we compare automata in figures 1 and 2, obviously the first one is more understandable than the second one. Does it exist a method to remove complex expressions is arcs and states, possibly adding more states?

– The multiplicity is important in learning. Automata in figures 3 and 4 show that, when the multiplicity doesn't care, smaller automata can be obtained. How much can this reduction be? Any learnable function remains learnable if the multiplicity is not taken into account?

## References

[Ang88] Angluin, D.: Queries and concept learning. Machine Learning 2(4), 319–342 (1988)

[BBB+96] Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: On the applications of multiplicity automata in learning. In: IEEE Symposium on Foundations of Computer Science, pp. 349–358 (1996)

[BBB+00] Beimel, A., Bergadano, F., Bshouty, N.H., Kushilevitz, E., Varricchio, S.: Learning functions represented as multiplicity automata. J. ACM 47(3), 506–530 (2000)

[CP71] Carlyle, J.W., Paz, A.: Realizations by stochastic finite automaton. J. Comput. Syst. Sci. 5, 26–40 (1971)

[Fli74] Fliess, M.: Matrices de hankel. J. Math. Pures Appl. 53, 197–222 (1974) (Erratum in vol. 54, 1975)

[Vil96] Vilar, J.M.: Query learning of subsequential transducers. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 72–83. Springer, Heidelberg (1996)

# Towards Feasible PAC-Learning of Probabilistic Deterministic Finite Automata*

Jorge Castro and Ricard Gavaldà

Departament de Llenguatges i Sistemes Informàtics
LARCA Research Group
Universitat Politècnica de Catalunya, Barcelona
{castro,gavalda}@lsi.upc.edu

**Abstract.** We present an improvement of an algorithm due to Clark and Thollard (Journal of Machine Learning Research, 2004) for PAC-learning distributions generated by Probabilistic Deterministic Finite Automata (PDFA). Our algorithm is an attempt to keep the rigorous guarantees of the original one but use sample sizes that are not as astronomical as predicted by the theory. We prove that indeed our algorithm PAC-learns in a stronger sense than the Clark-Thollard. We also perform very preliminary experiments: We show that on a few small targets (8-10 states) it requires only hundreds of examples to identify the target. We also test the algorithm on a web logfile recording about a hundred thousand sessions from an ecommerce site, from which it is able to extract some nontrivial structure in the form of a PDFA with 30-50 states. An additional feature, in fact partly explaining the reduction in sample size, is that our algorithm does not need as input any information about the distinguishability of the target.

## 1 Introduction

### 1.1 Context

Probabilistic Finite-State Automata (PFA) are thoroughly studied objects, both because of its inherent theoretical interest and their applications. Probabilistic Deterministic Finite-State Automata (PDFA) are a robust and natural subclass of PFA: See [6] for a study of the relations among these models, as well as HMM and POMDP.

These devices generate distributions on strings, and learning to approximate them from a sample is one of the central associated problems. A good number of algorithms have been proposed to infer PDFA. Some of them are only empirically evaluated while, for others, convergence in the limit to the target PDFA can be proven; see among others [1, 4, 2, 15, 11].

* Research supported in part by the EU PASCAL2 Network of Excellence and by the Spanish Ministry of Education and Science under projects MOISES-TA and TRANGRAM.

In the more demanding PAC model, some evidence that learning PDFA is hard was provided by Kearns *et al.* [10]. More precisely, it is shown in [10] that assuming that noisy parities are hard to PAC-learn, distributions generated by 2-letter PDFA cannot be PAC-learned in time polynomial in $n$, $1/\epsilon$, and $1/\delta$, where from now on $n$ denotes an upper bound on the number of states in the target machine, and $\epsilon$ and $\delta$ are the usual accuracy and confidence parameters in the PAC framework.

On the other hand, Ron *et al.* [13] gave an algorithm to PAC learn acyclic PDFA if polynomiality is measured in an additional parameter, the *distinguishability* of the target states - which we will denote as $\mu$ from now on. This formalized the observation, present already e.g. in [2], that one of the reasons that made some PDFA hard to learn was the presence of states with very similar suffix distributions. Clark and Thollard [3] showed how to extend this result to cyclic PDFA if still another parameter, the expected length of the generated strings $L$ is taken into account. Their work is the culmination of a line of research, in the sense of identifying a set of parameters that make polynomial-time learning possible. We will state their result precisely in Section 2.

A number of papers have since presented variations or extensions of Clark and Thollard's algorithm (for brevity, called the C-T algorithm from now on). The related paper [14] by the same authors presents a more algorithmic view of the same ideas, with emphasis on the structure identification part. Palmer and Goldberg [12] showed the analogous result for learning with respect to the variation ($L_1$) distance rather than the KL-distance as C-T. Guttman *et al* [9] showed that the class of $\mu_2$-distinguishable PDFA is also learnable with respect to the KL-distance; C-T uses the easier $\mu_\infty$-distinguishability measure. See also the related results in Guttman's thesis [8]. Denis *et al* [5] gave a quite deep PAC-style result for the full class of PFA, although the parameters in which the algorithm is polynomial are not completely identified there. Gavaldà *et al.* [7] give another variation of C-T that adapts to the complexity of the target, in the sense that it may stop earlier than the worst-case bound if convergence is achieved.

While the Clark-Thollard result proves polynomial-time learnability of PDFA, the actual polynomial is huge for interesting parameter values. For example, it is in the order of $10^{24}$ for $|\Sigma| = 2$, $n = L = 6$, and $\epsilon = \delta = \mu = 0.1$. For values similar to these ones, the algorithm in [7] uses sample sizes in the order of $10^5$ in their experiments. On the other hand, these algorithms do not look that different from other state-merging algorithms in the literature, which often do pretty well in practice with much smaller samples.

We believe that it is an interesting question whether these huge numbers are unavoidable if PAC-like guarantees are required or whether one can design algorithms with guaranteed polynomiality in well-defined parameters that have use realistic sample sizes; or, let us say, about as realistic as those of the algorithms which have been validated empirically only. It is important to note that we discuss algorithms having no prior knowledge about the structure of the state space; otherwise, the problem is much simpler.

## 1.2 Our Results

Our initial intention in this work was to produce an algorithm that does not ask for a bound on the distinguishability $\mu$ of the target PDFA. This value is in practice very hard to guess, and basically only trial-and-error can be used. As the work progressed, we incorporated other optimizations, all of which can still be rigorously justified. Yet, our algorithm is as easy to describe, if not more, than the original C-T algorithm.

We show that our algorithm PAC-learns in the same sense as that the C-T algorithm. In fact it learns with respect to a more demanding notion of distinguishability than the $L_\infty$-distance as C-T, which we call pref$L_\infty$-distance. This proof is the core of the paper.

While all our improvements are technical rather than conceptual, their combination could lead to dramatic experimental speedups. We describe a few experiments with an implementation of our algorithm that, we admit, are still far from being "an experimental evaluation". We first use the example PFAs in [7], having 10 states each, for which the algorithm in [7] required about $4 \cdot 10^5$ examples to converge. Our algorithm identifies the structure of the PDFAs and achieves low error with about 200-500 examples, i.e., a reduction by a factor of 1000 w.r.t. [7]. An additional example taken from [2] produces similar results.

We perform an additional experiment on a large dataset: a weblog of a high-traffic Spanish online travel agent, recording about 120,000 user sessions. Each session can be modelled as a string over an alphabet of size about 90, and average length about 12. On this dataset, our algorithm is able to identify some nontrivial structure: it extracts PDFA with 30-50 states that are certainly quite different from trees. We are currently in touch with the company to assess whether the patterns embodied in the PDFA make sense to local experts.

To finish this section, let us remark an important difference of our algorithm with C-T, of a high-level rather than purely technical nature: The C-T algorithm receives a number of parameters of the target as input, computes the worst-case number of examples required for those parameters, and asks for the full sample of that size upfront. Rather, we place ourselves in the more common situation where we have a given, fixed sample and we have to extract as much information as possible from it. Our main theorem then says that the algorithm PAC-learns provided this sample is large enough with respect to the target's parameters (some of which, such as $\mu$, are unknown; we are currently working on removing the need to have the other parameters as inputs).

## 2 Preliminaries

We essentially follow notation in [3]. A PDFA $A$ is a tuple $\langle Q, \Sigma, \tau, \gamma, \xi, q_0 \rangle$ where $Q$ is a finite set of states, $\Sigma$ is the alphabet, $\tau : Q \times \Sigma \longrightarrow Q$ is the transition function, $\gamma : Q \times (\Sigma \cup \{\xi\}) \longrightarrow [0,1]$ defines the probability of emitting each symbol from each state ($\gamma(q, \sigma) = 0$ when $\sigma \in \Sigma$ and $\tau(q, \sigma)$ is not defined), $\xi$ is a special symbol not in $\Sigma$ reserved to mark the end of a string, and $q_0 \in Q$ is the initial state. Transition function $\tau$ is extended to $Q \times \Sigma^\star$ in the usual way.

Given an observation string $x\xi = \sigma_0 \ldots \sigma_k \xi$ emitted by a known PDFA $A$, the state at each step can be tracked by starting from the initial state $q_0$ and following the labelled transitions according to $x$ until reaching last symbol $\xi$. Also, the probability of generating a given string $x\xi$ from state $q$ can be calculated recursively as follows: if $x$ is the empty word $\lambda$ the probability is $\gamma(q, \xi)$, otherwise $x$ is a string $\sigma_0\sigma_1 \ldots \sigma_k$ with $k \geq 0$ and $\gamma(q, \sigma_0\sigma_1 \ldots \sigma_k\xi) = \gamma(q, \sigma_0)\gamma(\tau(q, \sigma_0), \sigma_1 \ldots \sigma_k\xi)$.

The probability of state $q$ in PDFA $A$ is defined as the sum of values $\gamma(q_0, x\xi)$, where $x$ ranges over the set of strings in $\Sigma^\star$ that traverse $q$.

Assuming every state of $A$ has non-zero probability of generating some string, one can define for each state $q$ a probability distribution $D_q^A$ on $\Sigma^\star$: For each $x$, probability $D_q^A(x)$ is $\gamma(q, x\xi)$. The one corresponding to the initial state $D_{q_0}^A$ is called the distribution defined by $A$, written $D^A$ in short. When there is no ambiguity, we will omit superindex $A$.

Given a multiset $S$ of strings from $\Sigma^\star$ we denote by $S(x)$ the multiplicity of $x$ in $S$, write $|S| = \sum_{x\in\Sigma^\star} S(x)$ and for every $\sigma \in \Sigma$ define $S(\sigma) = \sum_{x\in\Sigma^\star} S(\sigma x)$. To resolve the ambiguity of this notation on strings of length 1, we will always use greek letters to mean elements of $\Sigma$, and latin letters for strings. We also denote by $S(\xi)$ the multiplicity of the empty word, $S(\lambda)$. To each multiset $S$ corresponds an empirical distribution $\hat{S}$ defined in the usual way, $\hat{S}(x) = S(x)/|S|$. Finally, prefixes$(S)$ denotes the multiset of prefixes of strings in $S$.

We consider several measures of divergence between distributions. Let $D_1$ and $D_2$ be probability distributions on $\Sigma^\star$. The Kullback–Leibler divergence, KL for short, is defined as

$$\mathrm{KL}(D_1, D_2) = \sum_x D_1(x) \log \frac{D_1(x)}{D_2(x)}.$$

The $L_\infty$ supremum distance is

$$L_\infty(D_1, D_2) = \max_{x\in\Sigma^\star} |D_1(x) - D_2(x)|.$$

Finally, we also use the supremum distance on prefixes (introduced here, as far as we know):

$$\mathrm{pref}L_\infty(D_1, D_2) = \max_{x\in\Sigma^\star} |D_1(x\Sigma^\star) - D_2(x\Sigma^\star)|.$$

**Definition 1.** *We say distributions $D_1$ and $D_2$ are* $\mu$-distinguishable *when $\mu \leq \max\{L_\infty(D_1, D_2), \mathrm{pref}L_\infty(D_1, D_2)\}$. A PDFA $A$ is $\mu$-distinguishable when for each pair of states $q_1$ and $q_2$ their corresponding distributions $D_{q_1}$ and $D_{q_2}$ are $\mu$-distinguishable.*

Observe that $\mathrm{pref}L_\infty(D_{q_1}, D_{q_2}) \geq \mu$ iff there is any $x \in \Sigma^\star$ such that $|\gamma(q_1, x) - \gamma(q_2, x)| \geq \mu$. By definition, our measure of distinguishability is never smaller than the usual $L_\infty$-distinguishability in the literature [3, 12].

## 3 Description of the Algorithm

We show below a learning algorithm for PDFAs that has as input parameters the alphabet size $|\Sigma|$, an upper bound $L$ on the expected length of strings emitted from any state of the target (alternatively, a bound on the expected length of strings from the initial state and a bound on the variance), an upper bound $n$ on the number of states of the target, and the confidence ($\delta$) and precision ($\epsilon$) parameters. In contrast with the C-T algorithm, it does not need as input parameter the distinguishability $\mu$ of the target.

According to [3], a PAC learner for the class of PDFA can be easily obtained from a polynomial-time algorithm, so-called `Learner` from now on, satisfying the requirements listed below; we follow their notation.

1. `Learner` returns (with high probability) a graph $G$ isomorphic to a subgraph of the target PDFA $A$. This means that there is a bijection $\Phi$ from a subset of states of $A$ to all nodes of $G$ such that 1) $\Phi(q_0) = v_0$ (where $q_0$, $v_0$ are the initial states of $A$ and $G$, respectively) and 2) if $\tau_G(v, \sigma) = w$ then $\tau(\Phi^{-1}(v), \sigma) = \Phi^{-1}(w)$.
2. The states in $A$ whose probability is greater than $\epsilon_2/(L+1)$, which we call *frequent states*, have a representative in $G$. That is $\Phi$ is defined on frequent states.
3. If $q$ is a frequent state in $A$ and $\sigma \in \Sigma$ is such that $\gamma(q, \sigma) > \epsilon_5$ (we say $(q, \sigma)$ is a *frequent transition*) then $\tau_G(\Phi(q), \sigma)$ exists and it equals $\Phi(\tau(q, \sigma))$.
4. A multiset $S_v$ is attached to every node $v$ in the graph. If $v$ represents a frequent target state $q$ (i.e., $\Phi(q) = v$ where $q$ is frequent), then for every $\sigma \in \Sigma \cup \{\xi\}$, it holds $|S_v(\sigma)/|S_v| - \gamma(q, \sigma)| < \epsilon_1$. A multiset holding this property is said to be $\epsilon_1$-good.

Numbers $\epsilon_1$, $\epsilon_2$ and $\epsilon_5$ above and auxiliar quantities $\epsilon_0$ and $\delta_0$ that we use later are defined as follows. Note that they do not depend on $\mu$.

$$
\begin{aligned}
\epsilon_1 &= \frac{\epsilon^2}{16(|\Sigma|+1)(L+1)^2} \\
\epsilon_2 &= \frac{\epsilon}{4n(n+1)L(L+1)\log(4(L+1)(|\Sigma|+1)/\epsilon)} \\
\epsilon_5 &= \frac{\epsilon}{4|\Sigma|(n+1)L(L+1)\log(4(L+1)(|\Sigma|+1)/\epsilon)} \\
\epsilon_0 &= \frac{\epsilon_2\epsilon_5}{n|\Sigma|(L+1)} \\
\delta_0 &= \frac{\delta}{n^2|\Sigma| + 3n|\Sigma| + n}
\end{aligned}
$$

From a such graph $G$ a PDFA hypothesis $H$ can be easily built having a small KL divergence with respect to $A$. This is described in the paragraphs "Completing the Graph" and "Estimating Probabilities" in [3], page 480. Basically, it is enough to complete the graph when necessary by introducing a new node, the *ground node*, representing all the low frequency states and new transitions to the ground node. Finally, a smoothing scheme is performed in order to estimate the transition probabilities.

The proof that an algorithm `Learner` with these properties, plus this additional graph completion and probability estimation step, is a PAC-learner is essentially the contents of Sections 4.3, 4.4 and 5 in [3]. It does not involve distinguishability at all, so we can apply it in our setting even if we have changed our measure of distinguishability.

Our learning algorithm takes as inputs the parameters listed above and a sample from the target machine containing $N$ examples. `Learner` performs at most $n|\Sigma|+1$ learning stages, each one making a pass over all training examples and guaranteed to add one transition to the graph $G$ it is building.

At the beginning of each stage, `Learner` has a graph $G$ that summarizes our current knowledge of the target $A$. Nodes and edges in $G$ represent, respectively, states and transitions of the target $A$. We call *safe nodes* the nodes of $G$, as they are inductively guaranteed (with probability at least $1-\delta_0$) as stand for distinct states of $A$, with transitions among them as in $A$. Safe nodes are denoted by strings in $\Sigma^\star$.

Attached to each safe node $v$ there is a multiset $S_v$ that keeps information about the distribution on the target state represented by $v$. The algorithm starts with a trivial graph $G$ consisting of a single node $v_0 = \lambda$ representing the initial state $q_0$ of the target, whose attached multiset is formed by all the available examples.

When a new stage starts, the learner adds a candidate node $u = v_u\sigma$ for each (safe) node $v_u$ of $G$ and each $\sigma \in \Sigma$ such that $\tau_G(v_u, \sigma)$ is undefined. Candidate nodes gather information about transitions of $A$ leading from states that have already a safe representative in $G$ but not having yet an edge counterpart in $G$. Attached to each candidate node $u$ there is also a multiset $S_u$, initially empty. The learner also keeps, for each candidate node $u$, a list $L_u$ of safe nodes that have not been yet distinguished (proved different) from $u$. Initially $L_u$ contains all nodes in $G$.

For each training example $x\xi = \sigma_0 \dots \sigma_{i-1}\sigma_i\sigma_{i+1} \dots \sigma_k\xi$ in the dataset, `Learner` traverses the graph matching each observation $\sigma_i$ to a state until either (1) all observations in $x$ have been exhausted or (2) a transition to a candidate node is reached. This occurs when all transitions up to $\sigma_{i-1}$ are defined and lead to a safe node $w$, but there is no edge out of $w$ labeled by $\sigma_i$. In this case, we add the suffix $\sigma_{i+1} \dots \sigma_k$ to the multiset $S_u$ of candidate node $u = w\sigma_i$ and, before processing a new example, we consider all pairs $(u, v)$ where safe node $v$ belongs to $L_u$. For each such pair, we call function `Test_Distinct`$(u, v)$ described in Figure 1. If the test returns “distinct”, we assume that $u$ and $v$ reach distinct states in the target and we remove $v$ from $L_u$.

A candidate node becomes *important* when $|S_u|$ exceeds $\epsilon_0 N/2$. Every time that there is an important candidate node $u = v_u\sigma$ whose associated set $L_u$ is empty, $u$ is promoted to a new safe node ($G$ gets a new node labelled with $u$), $u$ is not anymore a candidate node, and an edge from $v_u$ to $u$ labeled by $\sigma$ is added to $G$. The multiset $S_u$ is attached to the new safe node, $u$ is included in the list $L_{u'}$ of all remaining candidate nodes $u'$, and the phase continues.

If all training examples are processed without the condition above occurring, the algorithm checks whether there are any important candidate nodes left. If none remains, `Learner` returns $G$ and stops. Otherwise, it closes the phase as follows: It chooses the important candidate $u = v_u\sigma$ and the safe node $v \in L_u$ having smallest distinguishability on the empirical distributions (samples), and identifies them, by adding to $G$ an edge from $v_u$ to $v$ labeled by $\sigma$.

Finally, the phase ends by erasing all the candidate nodes and another phase starts.

```
function Test_Distinct($u, v$)
    //$u$ is a candidate node; $v$ is safe
    $m_u \leftarrow |S_u|$; $s_u \leftarrow |\,\text{prefixes}(S_u)|$;
    $m_v \leftarrow |S_v|$; $s_u \leftarrow |\,\text{prefixes}(S_v)|$;
    $t_{u,v} \leftarrow \left(\frac{2}{\min(m_u,m_v)} \ln \frac{4m_u^2(s_u+s_v)\pi^2}{3\delta_0}\right)^{1/2}$
    $d \leftarrow \max\left(L_\infty(\hat{S}_u,\hat{S}_v), \text{pref}L_\infty(\hat{S}_u,\hat{S}_v)\right)$
    if $d > t_{u,v}$ then return “distinct”
                  else return “not clear”
```

**Fig. 1.** The state-distinctness test

## 4 Analysis

In this section we show that algorithm `Learner` satisfies conditions (1)-(4) before, and therefore can be turned into a PAC-learner for PDFA.

The following two lemmas describe the behavior of `Test_Distinct`. Here $D_u$ (respectively, $D_v$) denotes the target distribution on state $q = \tau(q_0, u)$ ($q = \tau(q_0, v)$).

**Lemma 2.** *If* $D_u = D_v$ *function* `Test_Distinct`$(u, v)$ *returns “not clear” with probability* $1 - 6\delta_0/(\pi^2 m_u^2)$*.*

*Proof.* Let $D = D_u (= D_v)$. Function `Test_Distinct` returns “different” when there exists a string $x$ such that $|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| > t_{u,v}$ or $|\hat{S}_u(x) - \hat{S}_v(x)| > t_{u,v}$. First, we bound the probability of the event $\text{pref}L_\infty(\hat{S}_u, \hat{S}_v) > t_{u,v}$. To avoid summing over infinitely many $x$, consider $S_u \cup S_v$ ordered, say lexicographically. Then the event above is equivalently to saying “there is an index $i$ in this ordering such that some prefix of the $i$th string in $S_u \cup S_v$ in this ordering, call it $x$, satisfies the condition above”. (This is another way of saying that only $x$’s appearing in prefixes$(S_u \cup S_v)$ can make the inequality true, since all others have $\hat{S}_u(x\Sigma^\star) = \hat{S}_v(x\Sigma^\star) = 0$.) Therefore, its probability is bounded above by the maximum of $(s_u + s_v)\Pr[\,|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| > t_{u,v}]$ over all strings $x$. By the triangle inequality, this is at most

$$(s_u+s_v)\left(\Pr[\,|\hat{S}_u(x\Sigma^\star) - D(x\Sigma^\star)| > t_{u,v}/2] + \Pr[|\hat{S}_v(x) - D(x\Sigma^\star)| > t_{u,v}/2]\right).$$

Since $E[\hat{S}_u(x\Sigma^\star)] = E[\hat{S}_v(x\Sigma^\star)] = D(x\Sigma^\star)$, by Hoeffding's inequality this is at most

$$\begin{aligned}&(s_u + s_v)(2\,\exp(-2(t_{u,v}^2/4)\,m_u) + 2\,\exp(-2(t_{u,v}^2/4)\,m_u))\\ &\leq 4(s_u+s_v)\exp(-(t_{u,v}^2/2)\,\min(m_u,m_v)),\end{aligned}$$

which is $3\delta_0/(\pi^2 m_u^2)$ by definition of $t_{u,v}$.

A similar reasoning also shows that the probability of the event $L_\infty(\hat{S}_u, \hat{S}_v) > t_{u,v}$ is at most $3\delta_0/(\pi^2 m_u^2)$ and we are done.

**Lemma 3.** *If $D_u$ and $D_v$ are $\mu$-distinguishable and $\min(m_u, m_v) \geq \frac{8}{\mu^2} \ln \frac{8(m_u+m_v)m_u^2 L\pi^2}{3\delta_0^2}$ then* `Test_Distinct`$(u, v)$ *returns "different" with probability $1 - \delta_0$.*

*Proof.* We first bound the size of prefixes$(S_u \cup S_v)$. Clearly, its expected size is at most $L|S_u \cup S_v|$. Then, by Markov's inequality, $\Pr[|\,\text{prefixes}(S_u \cup S_v)| \geq \frac{2}{\delta_0} \cdot L|S_u \cup S_v|]$ is less than $\delta_0/2$. Therefore, we have with probability at least $1 - \delta_0/2$ that $s_u + s_v \leq 2(m_u + m_v)L/\delta_0$.

Now assume there is a string $x$ witnessing that pref$L_\infty(D_u, D_v) > \mu$ (otherwise some $x$ is a witness for $L_\infty(D_u, D_v) > \mu$ and we argue in a similar way), i.e. a string such that $|D_u(x\Sigma^\star) - D_v(x\Sigma^\star)| > \mu$. If $\min(m_u, m_v) \geq \frac{8}{\mu^2} \ln \frac{8(m_u+m_v)m_u^2 L\pi^2}{3\delta_0^2}$, by the argument above with high probability we have $t_{u,v} \leq \mu/2$ and the probability of returning "different" is at least the probability of the event $|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| > \mu/2$. The hypothesis on $x$ and the triangle inequality shows that probability of the complementary event $|\hat{S}_u(x\Sigma^\star) - \hat{S}_v(x\Sigma^\star)| \leq \mu/2$ is at most $\Pr[\,|\hat{S}_u(x\Sigma^\star) - D_u(x\Sigma^\star)| > \mu/4] + \Pr[\,|\hat{S}_v(x\Sigma^\star) - D_v(x\Sigma^\star)| > \mu/4]$. By the Hoeffding bound, this sum is less than $\delta_0/2$, and we are done.

Lemmas 4–8 below share the hypothesis the current $G$ is isomorphic to a subgraph of $A$ and deal with one fixed stage of the learning algorithm. Probabilities are taken over samples.

**Lemma 4.** *Let $u$ be a candidate node. If $u$ is promoted to safe (in this stage) then, with probability $1 - \delta_0$, node $u$ corresponds to a new target state, i.e., one not represented in the current graph $G$.*

*Proof.* We show that a candidate node $u$ representing the same target state than a safe state $v$ has very small probability of being promoted. By Lemma 2 at any specific call to `Test_Distinct`$(u, v)$, the function returns the wrong value "different" with probability at most $6\delta_0/(\pi^2 m_u^2)$. The test is called once for every example included in $S_u$, so the value of $m_u$ increases by 1 at each consecutive call within the stage. Therefore, the probability that safe node $v$ is ruled out from $L_u$ is at most

$$\sum_{m_u \geq 1} 6\delta_0/(\pi^2\, m_u^2) = \delta_0$$

**Lemma 5.** *Let $u$ be a candidate node, and let $\mu$ be the distinguishability of the target. If $N$ is greater than $\frac{16}{\epsilon_0 \mu^2}(3e \ln \frac{48}{\epsilon_0 \mu^2} + \ln \frac{16 L\pi^2}{3\delta_0^2})$ with probability $1 - n\delta_0$, if* *candidate node $u$ is merged with a safe node $v$ then, strings $u$ and $v$ end in the same state in the target.*

*Proof.* Assume candidate $u$ is merged with safe node $v$. Necessarily $u$ is important and $N \geq m_u > \epsilon_0 N/2$. As $v$ is safe it also holds $m_v > \epsilon_0 N/2$. It is also clear that $m_u + m_v \leq 2N$. From these values of $m_u$, $m_v$ and the hypothesis on $N$, it can be checked that $\min(m_u, m_v)$ satisfies requirement in Lemma 3. So, if they were representatives of different target states, safe $v$ would remain in $L_u$ with probability at most $\delta_0$.

**Lemma 6.** *Assume that, after processing all examples, graph $G$ has no safe node $v$ representing a frequent state $q$ of $A$. If $N > \frac{8(L+1)}{\epsilon_2} \ln \frac{1}{\delta_0}$ then, with probability $1 - \delta_0$, the learner will not finish yet.*

**Lemma 7.** *Assume that, after processing all examples, graph $G$ has no edge representing a frequent transition $(q, \sigma)$ in $A$. If $N > \frac{8(L+1)}{\epsilon_2 \epsilon_5} \ln \frac{1}{\delta_0}$ then with probability $1 - \delta_0$, some candidate is important and the learner will not finish yet.*

**Lemma 8.** *Let $u$ be a candidate node. If the number $N$ of training examples is greater than $\frac{1}{\epsilon_0 \epsilon_1^2} \ln \frac{2(|\Sigma|+1)}{\delta_0}$ and $u$ is promoted to safe then, with probability $1 - \delta_0$, multiset $S_u$ is $\epsilon_1$-good.*

Let $N_0$ be $\max\left(\frac{16}{\epsilon_0 \mu^2}(3e \ln \frac{48}{\epsilon_0 \mu^2} + \ln \frac{16 L \pi^2}{3 \delta_0^2}), \frac{8}{\epsilon_0 \epsilon_1^2} \ln \frac{2(|\Sigma|+1)}{\delta_0}\right)$. A straightforward induction shows the main theorem:

**Theorem 9.** *If $N > N_0$, with probability $1 - \delta$* `Learner` *returns a graph $G$ satisfying requirements (1)–(4) listed above.*

As explained already, the second phase of the learning algorithm takes the graph $G$, completes it if necessary, and sets the transition probabilities according to their empirical distribution. An additional smoothing is performed, as described in [3]. We do not describe it here, but state that the resulting PDFA will approximate the target in the KL distance, as can be deduced from the proof in [3].

## 5 Experiments

### 5.1 Small Targets

Our first experiments used the two automata tested by Gavaldà *et al.* [7], shown in Figure 2. The one on the left is a (nondeterministic) HMM repeatedly generating strings in $\{abb, aaa, bba\}$ with different probabilities. The one on the right is the "cheese maze": at each state (or square), an observation (a letter in $\{1, 2, 3\}$) indicates the number of walls around that state, with the exception of $s10$ where the automaton terminates. They have thus 10 states each. We have additionally used the Reber grammar automaton, with 8 states, discussed in [2] and shown in Figure 3.

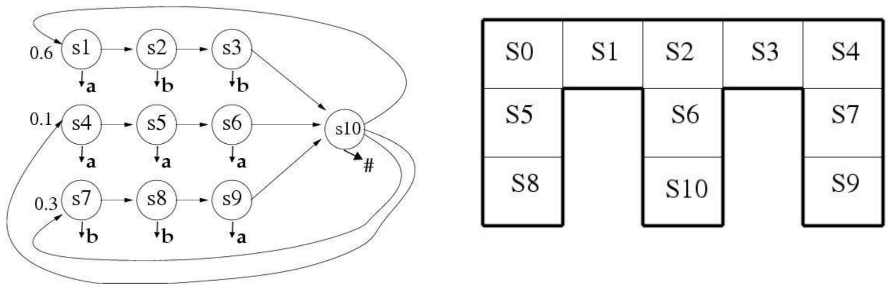


**Fig. 2.** Example PFAs from [7]

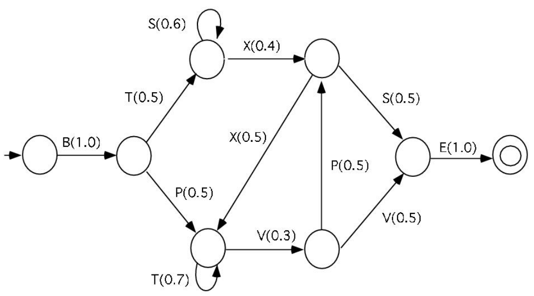


**Fig. 3.** The Reber grammar; plot taken from [2]

For each of these automata, and different values of $N$, we generated 10 examples of size $N$ from the target, and run our algorithm on these examples. In all experiments we used $\delta = 0.05$ and the (known) number of target states for $n$. For the Reber grammar, the full structure of the automaton was identified about half the times with $N = 100$, but systematically identified when $N = 200$, at which point transition probabilities were correct within (absolute) 5%. For $N = 1000$, transition probabilities were correct within 1%. For the cheese maze automaton, at $N = 300$ the structure was found 9 out of 10 times, with transition probabilities correct up to 2%. For $N = 1000$, the structure was correctly found in all trials. Interestingly, when the program was changed to use only $L_\infty$-, rather than pref$L_\infty$-distinguishability, the point at which the structure is identified more than 50% of the times was around $N = 1300$. That is, using pref$L_\infty$ did help in this case. Results were similar for the HMM on the left of Figure 2.

## 5.2 An Experiment with a Real, Large Dataset

As a larger test, we used a web logfile recording sessions from a high-traffic Spanish online travel agency, selling flights, hotel stays, car rentals, and theater tickets. Each entry in the logfile records a user request to the company's web to initiate some action. The local experts distinguish 91 types of requests or

| Sample | # states | $L_1$ distance |
|---|---|---|
| 40k | 35 | .582 |
| 50k | 36 | .546 |
| 60k | 39 | .518 |
| 70k | 42 | .516 |
| 80k | 45 | .480 |
| 100k | 54 | .439 |

**Fig. 4.** Results on the online travel agency dataset

tags; some examples could be "search_flight", "search_hotel", "book_flight", "credit_card_info", "home", "register", "help", etc. We preprocessed the logfile transforming each request into a tag identifier and grouping clicks from the same user into sessions. Therefore each session can be viewed as a string over a 91-letter alphabet. The median and average of session length are 4 and 11.9, excluding 1-click sessions, and we had 120,000 sessions to work with.

We ran our algorithm on subsets of several sizes $N$ of this dataset. Since human web users cannot be perfectly modelled by PDFA, one should not expect high accuracy. Still, there are certainly patterns that users tend to follow, so it was worth checking whether any structure is found.

We tried $N = 40,000$ to $N = 100,000$ in multiples of $10,000$. Figure 4 presents, for each $N$, the size of the resulting PDFA and the $L_1$-distance from the dataset to a randomly generated sample of size $100,000$ from the output machine.

Note that the $L_1$ distance can have value up to 2. (In fact, we tried generating several independent samples of size 100k from the PDFA obtained with the 100k sample and computing their $L_1$ mutual distance. The results were around 0.39 even though they came from the same machine, so this value is really the baseline once sample size has been fixed to 100k.) The table shows that convergence to a fixed machine did not occur, which is no surprise. On the other hand, the resulting machines were not at all tree-like PDFAs that occur when no states can be merged: most safe states did absorb candidate states at some point. Given the alphabet size (91) and number of states ($\geq 30$), depicting and understanding them is not immediate.

Note that we have not discussed the values of $\epsilon$ and $L$ used in the experiments. In fact, our implementation does not use them: they are used only to determine when a state is important. In particular, observe that $\epsilon$ and $L$ are not used in the state distinctness test. The implementation keeps merging candidate states as long as there is any left at the end of the stage, hence every candidate state is eventually merged. We believe that it is possible to prove that this variant is still a PAC learner, since non-important states, after smoothing, by definition do not contribute much to the KL distance. We believe it is also possible to remove the need for an upper bound on $n$ without significantly increasing sample size in practice; this would give a PAC-learning whose only parameter is the confidence $\delta$.

## 6 Conclusions

We believe that these first experiments, as preliminary as they are, show that maybe one cannot rule out the existence of a provably-PAC learner that has reasonable sample sizes in practice. More systematic experimentation, as well as improving of our slow, quick-and-dirty implementation, is work in progress.

## References

[1] Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: ICGI, pp. 139–152 (1994)
[2] Carrasco, R.C., Oncina, J.: Learning deterministic regular grammars from stochastic samples in polynomial time. ITA 33(1), 1–20 (1999)
[3] Clark, A., Thollard, F.: PAC-learnability of probabilistic deterministic finite state automata. Journal of Machine Learning Research 5, 473–497 (2004)
[4] de la Higuera, C., Oncina, J., Vidal, E.: Identification of DFA: data-dependent vs data-independent algorithms. In: ICGI, pp. 313–325 (1996)
[5] Denis, F., Esposito, Y., Habrard, A.: Learning rational stochastic languages. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS (LNAI), vol. 4005, pp. 274–288. Springer, Heidelberg (2006)
[6] Dupont, P., Denis, F., Esposito, Y.: Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. Pattern Recognition 38, 1349–1371 (2005)
[7] Gavaldà, R., Keller, P.W., Pineau, J., Precup, D.: PAC-learning of Markov models with hidden state. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 150–161. Springer, Heidelberg (2006)
[8] Guttman, O.: Probabilistic Automata Distributions over Sequences. Ph.D. thesis, The Australian National University (September 2006)
[9] Guttman, O., Vishwanathan, S.V.N., Williamson, R.C.: Learnability of probabilistic automata via oracles. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 171–182. Springer, Heidelberg (2005)
[10] Kearns, M.J., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R.E., Sellie, L.: On the learnability of discrete distributions. In: STOC, pp. 273–282 (1994)
[11] Kermorvant, C., Dupont, P.: Stochastic grammatical inference with multinomial tests. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 149–160. Springer, Heidelberg (2002)
[12] Palmer, N., Goldberg, P.W.: PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 157–170. Springer, Heidelberg (2005)
[13] Ron, D., Singer, Y., Tishby, N.: On the learnability and usage of acyclic probabilistic finite automata. J. Comput. Syst. Sci. 56(2), 133–152 (1998)
[14] Thollard, F., Clark, A.: Learning stochastic deterministic regular languages. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 248–259. Springer, Heidelberg (2004)
[15] Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: ICML, pp. 975–982 (2000)

# Learning Context-Sensitive Languages from Linear Structural Information[⋆]

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
jsempere@dsic.upv.es

**Abstract.** In this work we propose a method to infer context-sensitive languages from positive structural examples produced by linear grammars. Our approach is based on a representation theorem induced by two operations over strings: duplication and reversal. The inference method produces an acceptor device which is an unconventional model of computation based on biomolecules (*DNA computing*). We prove that a subclass of context-sensitive languages can be inferred by using the representation result in combination with reductions from linear languages to $k$-testable in the strict sense regular languages.

**Keywords:** context-sensitive languages, Watson-Crick finite automata, linear languages, $k$-testable languages, identifiability from positive structural data.

## 1 Introduction

In the recent times, an unconventional theory of computation based on some biomolecules behavior has been proposed as the research area of DNA computing [9]. In a wide point of view, DNA computing deals with the capacities of DNA molecules to make (universal) computations. So, different models have been proposed in the framework of the formal language theory with some ingredients of the DNA features in order to process strings. We can mention Watson-Crick finite automata, sticker systems, splicing systems, Insertion-Deletion systems, among others. A profound study of these new models has pointed out its capacity to characterize language classes from Chomsky's hierarchy and new language classes which are related to the previous ones. In addition, these new models have provided a new look to the formal language theory in the sense that they have provided new operations over strings (splicing, duplication, twin shuffles, etc.) and new representations for the languages (i.e. circular strings or double strings). A comprehensive reference in this field is [9].

In this work, we will work with a DNA based computing model, the Watson-Crick finite automaton (WKFA) [3]. It has been proved that this model is able

[⋆] Work supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769.

to recognize context-sensitive languages. In addition, the languages accepted by WKFA can be obtained as the intersection between linear languages and even linear ones. So, given that these language families have been widely studied in the framework of Grammatical Inference we can take some advantages of previous results in order to learn efficiently some language classes which are characterized by restricted versions of WKFA. Observe that this approach enables the inference of new language classes which contains non-trivial context-sensitive languages.

The structure of this work is as follows: First we will give basic definitions and we will fix the notation related to formal language theory and some aspects of DNA computing used in the sequel. In section 3, we will propose an algorithm to learn language classes characterized by some restricted versions of the WKFA. We will prove the identifiability in the limit of these new classes based on the reducibility of this problem to previously solved ones. We will generalize our results by providing a learning scheme for different language classes. Finally, we will show our conclusions and we will provide some future research guidelines.

## 2 Basic Concepts and Notation

In this section, we will provide some concepts from formal language theory and DNA computing models. We suggest the books [9] and [10] to the reader.

### Formal Languages

An alphabet $\Sigma$ is a finite non-empty set of elements named symbols. A string defined over $\Sigma$ is a finite ordered sequence of symbols from $\Sigma$. The infinite set of all the strings defined over $\Sigma$ will be denoted by $\Sigma^*$. Given a string $x \in \Sigma^*$ we will denote its length by $|x|$. The set of strings defined over $\Sigma$ with length equals to (less than) $k$ will be denoted by $\Sigma^k$ ($\Sigma^{\leq k}$). The empty string will be denoted by $\lambda$ and $\Sigma^+$ will denote $\Sigma^* - \{\lambda\}$. Given a string $x$ we will denote by $x^r$ the reversal string of $x$. A language $L$ defined over $\Sigma$ is a set of strings from $\Sigma^*$.

A grammar is a construct $G = (N, \Sigma, P, S)$ where $N$ and $\Sigma$ are the alphabets of auxiliary and terminal symbols with $N \cap \Sigma = \emptyset$, $S \in N$ is the *axiom* of the grammar and $P$ is a finite set of productions in the form $\alpha \to \beta$. We will say that $w_1$ directly derives to $w_2$, and we will denote it by $w_1 \underset{G}{\Rightarrow} w_2$ if $w_1 = u\alpha v$, $w_2 = u\beta v$ and $\alpha \to \beta \in P$. We will denote by $\overset{*}{\underset{G}{\Rightarrow}}$ the reflexive and transitive closure of $\underset{G}{\Rightarrow}$. The language of the grammar is denoted by $L(G)$ and it is the set of terminal strings that can be obtained from $S$ by applying symbol substitutions according to $P$. So, $L(G) = \{w \in \Sigma^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

We will say that a grammar $G = (N, \Sigma, P, S)$ is *right linear* (regular) if every production in $P$ is in the form $A \to uB$ or $A \to w$ with $A, B \in N$ and $u, w \in \Sigma^*$. The class of languages generated by right linear grammars is the class of regular languages and will be denoted by $\mathcal{REG}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is *linear* if every production in $P$ is in the form $A \to uBv$ or $A \to w$ with $A, B \in N$ and $u, v, w \in \Sigma^*$. The class of languages generated by linear grammars will be denoted by $\mathcal{LIN}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is *even linear* if every production in $P$ is in the form $A \rightarrow uBv$ or $A \rightarrow w$ with $A, B \in N$, $u, v, w \in \Sigma^*$ and $|u| = |v|$. The class of languages generated by even linear grammars will be denoted by $\mathcal{ELIN}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is context-free if every production in $P$ is in the form $A \rightarrow w$ with $A \in N$ and $w \in (\Sigma \cup N)^*$. The class of languages generated by context-free grammars will be denoted by $\mathcal{CF}$.

A context-sensitive grammar is a grammar $G = (N, \Sigma, P, S)$ where every production in $P$ is in the form $\alpha A \beta \rightarrow \alpha \omega \beta$ with $\alpha, \beta \in (N \cup \Sigma)^*$, $\omega \in (N \cup \Sigma)^+$ and $A \in N$. If $S \rightarrow \lambda$ then $S$ does not appear in the right side of any other production in $P$. The class of the languages generated by context-sensitive grammars will be denoted by $\mathcal{CS}$. A well-known result from formal language theory is the inclusions $\mathcal{REG} \subset \mathcal{ELIN} \subset \mathcal{LIN} \subset \mathcal{CF} \subset \mathcal{CS}$.

A homomorphism $h$ is defined as a mapping $h : \Sigma \rightarrow \Gamma^*$ where $\Sigma$ and $\Gamma$ are alphabets. We can extend the definition of homomorphisms over strings as $h(\lambda) = \lambda$ and $h(ax) = h(a)h(x)$ with $a \in \Sigma$ and $x \in \Sigma^*$. The homomorphism over a language $L \subseteq \Sigma^*$ is defined as $h(L) = \{h(x) : x \in L\}$.

## ***Stickers*, Molecules and Watson-Crick Finite Automata**

Given an alphabet $\Sigma = \{a_1, \cdots, a_n\}$, we will use the symmetric (and injective) relation of complementarity $\rho \subseteq \Sigma \times \Sigma$. For any string $x \in \Sigma^*$, we will denote by $\rho(x)$ the string obtained by substituting the symbol $a$ in $x$ by the symbol $b$ such that $(a, b) \in \rho$ (remember that $\rho$ is injective) with $\rho(\lambda) = \lambda$.

Given an alphabet $\Sigma$, a *sticker* over $\Sigma$ will be the pair $(x, y)$ such that $x = x_1 v x_2$, $y = y_1 w y_2$ with $x, y \in \Sigma^*$ and $\rho(v) = w$. The sticker $(x, y)$ will be denoted by $\begin{pmatrix} x \\ y \end{pmatrix}$. A sticker $\begin{pmatrix} x \\ y \end{pmatrix}$ will be a *molecule* if $|x| = |y|$ and $\rho(x) = y$, and it will be denoted by $\begin{bmatrix} x \\ y \end{bmatrix}$. Obviously, any sticker $\begin{pmatrix} x \\ y \end{pmatrix}$ or molecule $\begin{bmatrix} x \\ y \end{bmatrix}$ can be represented by $x\#y^r$ where $\# \notin \Sigma$.

There have been different computational models and generating devices that use stickers and molecules to define formal languages. We will fix our attention to the acceptor models named *Watson-Crick finite automata*. A Watson-Crick finite automaton (WKFA) [3] is a good example of how DNA biological properties can be adapted to propose computation models in the framework of DNA computing. WK finite automata work with double strings inspired by double-stranded molecules with a complementary relation between elements, that is, the classical complementary relation between DNA nucleotides A-T and C-G. So, a WK finite automaton has an input double tape which is organized into upper and lower cells, it has two tape heads that access to the upper and lower cells and that can move independently and a finite control which holds a state of the machine during its computation. In addition, the automaton has a transition function that guides the movements of the machine. The machine works as follows: initially a sticker or molecule is placed in the double tape (i.e. the lower strand in the lower tape and the upper strand in the upper tape), the tape heads are placed at the beginning of the tape (i.e. pointing out to the first symbol of every tape) and the finite control holds an initial state. Then, the machine starts to apply the transition function which is nondeterministic. Every time that the machine applies a transition then the state in the finite control changes and the tape heads advance one cell to the right or stay at the same cell independently (i.e. maybe a transition only moves the upper head or only the lower head or it can move both heads). The machine stops when no transition can be applied or the input sticker or molecule have been completely processed. Observe that, in this case, the machine can halt into an acceptation state. The criterium which is imposed to accept a sticker is that it has been completely processed, the machine has stopped within an acceptation state and the sticker is a molecule.

Formally, an *arbitrary* WK finite automaton is defined by the tuple $M = (V, \rho, Q, s_0, F, \delta)$, where $Q$ and $V$ are disjoint alphabets (states and symbols), $s_0$ is the initial state, $F \subseteq Q$ is a set of final states and the finitely defined function $\delta : Q \times \begin{pmatrix} V^* \\ V^* \end{pmatrix} \rightarrow \mathcal{P}(Q)$ (which denotes the power set of $Q$, that is the set of all possible subsets of $Q$). Furthermore, we can impose a normal form such that for every transition $q \in \delta(q, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix})$ then $|x_1x_2| = 1$. This normal form defines the so called 1-*limited* WK finite automata and they were proved to be equivalent to arbitrary ones [3].

An instantaneous description of the WK finite automaton will be denoted by $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} q \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$, where $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ is the part of the sticker which has been processed, $q$ is the state of the finite control and $\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$ is the rest of the sticker to be processed. We can relate instantaneous descriptions as follows: $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} q \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1v_1 \\ y_1w_1 \end{pmatrix} p \begin{pmatrix} v_2 \\ w_2 \end{pmatrix}$ if $x_2 = v_1v_2$, $y_2 = w_1w_2$ and $p \in \delta(q, \begin{pmatrix} v_1 \\ w_1 \end{pmatrix}$. We will denote the reflexive and transitive closure of $\Rightarrow$ by $\overset{*}{\Rightarrow}$.

Given an arbitrary WK finite automaton $M = (V, \rho, Q, s_0, F, \delta)$, the language of molecules accepted by $M$ will be defined by the set $L_m(M) = \{ \begin{bmatrix} x \\ y \end{bmatrix} : s_0 \begin{bmatrix} x \\ y \end{bmatrix} \overset{*}{\Rightarrow} \begin{bmatrix} x \\ y \end{bmatrix} p$ with $p \in F\}$. The upper strand language accepted by $M$ will be defined by the set $L_u(M) = \{x : s_0 \begin{bmatrix} x \\ y \end{bmatrix} \overset{*}{\Rightarrow} \begin{bmatrix} x \\ y \end{bmatrix} p$ with $p \in F\}$. The family of upper languages accepted by arbitrary Watson-Crick finite automata will be denoted by $\mathcal{AWK}_u$, and it has been proved that $\mathcal{AWK}_u \subset \mathcal{CS}$. That is, WKFA accept context-sensitive languages in the upper strand. In addition, it has been proved that context-free languages and $\mathcal{AWK}_u$ are disjoint classes of languages [7].

In a previous work, we proved that the languages accepted by arbitrary WKFA can be represented by operations over linear and even linear languages [13]. The main theorem that supports this statement is the following

**Theorem 1.** *(Sempere, [13]) Let $M = (V, \rho, Q, s_0, F, \delta)$ be an arbitrary Watson-Crick finite automaton. Then, there exists a linear language $L_1$ and an even linear language $L_2$ such that $L_m(M) = L_1 \cap L_2$.*

In [13], we provided an algorithm to construct a linear grammar $G_1$ such that $L(G_1) = L_1$ and an even linear grammar $G_2$ such that $L(G_2) = L_2$. It works as follows: First, we can construct the grammar $G_1 = (N, V \cup \{\#\}, P, s_0)$ where $N = Q$, $s_0$ is the axiom of the grammar and $P$ is defined as follows

– If $q \in F$ then $q \rightarrow \# \in P$
– If $p \in \delta(q, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix})$ then $q \rightarrow x_1\ p\ x_2^r \in P$.

The language $L_2$ can be defined by the grammar $G_2 = (\{S\}, V \cup \{\#\}, P, S)$ where $P$ is defined as follows

– $S \rightarrow \# \in P$
– For every pair of symbols $a, b \in V$, such that $(a, b) \in \rho$, $S \rightarrow aSb \in P$

In order to characterize the upper strand language we provided the following result

**Corollary 1.** *(Sempere, [13]) Let $M = (V, \rho, Q, s_0, F, \delta)$ be an arbitrary WK finite automaton. Then $L_u(M)$ can be expressed as $g(h^{-1}(L_1 \cap L_2) \cap R)$ with $L_1$ being a linear language, $L_2$ an even linear language, $R$ a regular language and $g$ and $h$ two morphisms.*

Observe that the last result can be obtained by using a morphism $h : V \cup V' \cup \{\#\} \rightarrow V \cup \{\#\}$, defined as $h(a) = h(a') = a$ for every $a \in V$ where $V' = \{a' : a \in V\}$, and $h(\#) = \#$. Then, $R = V^*\#V'^*$ and $g$ is a morphism defined as $g(\#) = \lambda$, $g(a) = a$ and $g(a') = \lambda$ for every $a \in V$ and every $a' \in V'$.

*Example 1.* Let $M = (V, \rho, Q, q_0, F, \delta)$ be a WKFA where $V = \{a, b, c\}$, $\rho = \{(a, a), (b, b), (c, c)\}$, $F = \{q_f\}$ and $\delta$ is defined as follows:

$$\delta(q_0, \begin{pmatrix} a \\ \lambda \end{pmatrix}) = \{q_a\} \qquad \delta(q_a, \begin{pmatrix} a \\ \lambda \end{pmatrix}) = \{q_a\} \qquad \delta(q_a, \begin{pmatrix} b \\ a \end{pmatrix}) = \{q_b\}$$
$$\delta(q_b, \begin{pmatrix} b \\ a \end{pmatrix}) = \{q_b\} \qquad \delta(q_b, \begin{pmatrix} c \\ b \end{pmatrix}) = \{q_c\} \qquad \delta(q_c, \begin{pmatrix} c \\ b \end{pmatrix}) = \{q_c\}$$
$$\delta(q_c, \begin{pmatrix} \lambda \\ c \end{pmatrix}) = \{q_f\} \qquad \delta(q_f, \begin{pmatrix} \lambda \\ c \end{pmatrix}) = \{q_f\}$$

It can be easily proved that $L_u(M) = \{a^n b^n c^n : n \geq 1\}$. Then the corresponding linear grammar associated with $M$ is the following one, which we will name $G_1$:

$$q_0 \rightarrow aq_a \qquad q_a \rightarrow aq_a \mid bq_b a \qquad q_b \rightarrow bq_b a \mid cq_c b$$
$$q_c \rightarrow cq_c b \mid q_f c \qquad q_f \rightarrow q_f c \mid \#$$

The even linear grammar associated with $\rho$ is trivially defined by the rules $S \to aSa \mid bSb \mid cSc \mid \#$ which we will name $G_2$. Observe that $L_m(M) = L(G_1) \cap L(G_2)$ while the language $L_u(M)$ is obtained from the strings in $L(G_1) \cap L(G_2)$ by taking only the complete prefixes up to the # symbol. This last operation can be performed by applying Corollary 1.

From the previous results, it can be proved that upper strand languages accepted by WKFA can be reduced to regular languages. So, we introduced in the WKFA model well-known features such as *k-testability* [14] and *reversibility* [15]. In addition, we establish a way to obtain *regular-like* expressions from WKFA [16].

## Local Testability

Here, we will introduce the definition of local testability and local testability in the strict sense. For any string $x \in \Sigma^*$ and any integer value $k > 0$, the testability vector $v_k(x)$ is defined by the tuple $(i_k(x), t_k(x), f_k(x))$ where

$$i_k(x) = \begin{cases} x, & \text{if } |x| < k \\ u : x = uv, |u| = k-1 & \text{if } |x| \geq k \end{cases}$$

$$f_k(x) = \begin{cases} x, & \text{if } |x| < k \\ v : x = uv, |v| = k-1 & \text{if } |x| \geq k \end{cases}$$

$$t_k(x) = \{v : x = uvw, u, w \in \Sigma^* \wedge |v| = k\}$$

We will define the equivalence relation $\equiv_k$ in $\Sigma^* \times \Sigma^*$ as $x \equiv_k y$ iff $v_k(x) = v_k(y)$. It has been proved in [6] that $\equiv_k$ is a finite index relation and that $\equiv_k$ covers $\equiv_{k+1}$.

So, we will say that any language $L$ is $k$-testable iff it is defined as the union of some equivalence classes of $\equiv_k$. In addition, $L$ is local testable iff it is $k$-testable for any integer value $k > 0$. The family of $k$-testable languages will be denoted by $k - \mathcal{LT}$ while $\mathcal{LT}$ will denote the class of testable languages.

A different kind of testability is the so called testability in the strict sense which was again proposed in [6]. Here, for any alphabet $\Sigma$ we will take the sets $I_k, F_k \subseteq \Sigma^{\leq k-1}$ and $T_k \subseteq \Sigma^k$. Then, a language $L$ is said to be $k$-testable in the strict sense if the following equation holds

$$L \cap \Sigma^{k-1}\Sigma^* = (I_k\Sigma^*) \cap (\Sigma^* F_k) - (\Sigma^* T_k \Sigma^*).$$

Observe that, according to the last equation, any word in $L$ with length greater than or equals to $k-1$ begins with a segment in $I_k$, ends with a segment in $F_k$ and has no segment from $T_k$. Any language $L$ is locally testable in the strict sense iff it is $k$-testable in the strict sense for any $k > 0$. The family of $k$-testable languages in the strict sense will be denoted by $k - \mathcal{LTSS}$ while $\mathcal{LTSS}$ will denote the class of locally testable languages in the strict sense.

It has been proved that $k - \mathcal{LT}$ is the boolean closure of $k - \mathcal{LTSS}$ [22]. In addition, both classes $k - \mathcal{LT}$ and $k - \mathcal{LTSS}$ are subclasses of $\mathcal{REG}$.

## 3 Learning Watson-Crick Finite Automata from Positive Structural Data

In order to learn formal languages accepted by (restricted) WKFA we will use two operations to represent such languages: duplication and reversal. Once we fix this representation for formal languages, we can infer restricted versions of WKFA in order to recognize languages in the upper strand. Our method is a reduction technique based on the linear and even-linear languages used in Theorem 1 and Corollary 1. Observe, that a similar approach was employed in [8] where the authors reduced languages accepted by WKFA to regular languages by using a technique which is different from the one in [13].

The relation of complementarity that we will use is the trivial identity relation. That is, $(\forall a \in \Sigma)\ (a, a) \in \rho$. The duplication of a string is defined as $duplicate(x) = x\#x^r$. The extension over a set $S$ is trivially defined as $duplicate(S) = \{duplicate(x) : x \in S\}$. Observe that this is a linear time operation.

Our main task is to learn unknown linear languages from the strings obtained by duplication. Here, we can adopt different solutions in order to carry out the learning task. First, we can use learning algorithms for subclasses of linear languages as in [1,2,5]. Another option could be the use of structural information as in [5]. Observe that the use of structural information has been widely accepted as an information protocol since Sakakibara's works about learning context-free languages [11,12]. We will use structural information in this work in order to avoid the use of complete data (negative and positive strings). Nevertheless, the use of any algorithm to infer linear languages from different information protocols could be easily introduced in our learning scheme.

So, the information given to the learning algorithm will not be the duplicated strings but the structural information associated with them, according to an unknown WKFA.

Observe that the structural information from linear grammars allows the transformation to even linear ones as was shown in a previous work [17]. We can introduce the reduction in a formal manner as follows: Let us take the linear structured string $w$ defined over the alphabet $\Sigma$, then we can obtain an even linear structure from $w$ by applying the function $ell(w)$ with the following rules

1. $ell((\lambda)) = \lambda$
2. $ell((a)) = a$ for every $a \in \Sigma$
3. $ell((a(x))) = a \cdot ell((x)) \cdot *$ for every $a \in \Sigma$ and $x$ a structural string over $\Sigma$
4. $ell(((x)a)) = * \cdot ell((x)) \cdot a$ for every $a \in \Sigma$ and $x$ a structural string over $\Sigma$

The last transformation can be easily extended over sets of structural strings.

The even linear languages can be reduced to regular ones by using control sets as was proved in [21]. In addition, a different solution was proposed in [18] by using the $\sigma$ reduction over strings in $\Sigma^*$ that we will define as follows:

1. $\sigma(axb) = [ab]\sigma(x)$ with $a, b \in \Sigma$ and $x \in \Sigma^*$
2. $\sigma(a) = [a]$ with $a \in \Sigma \cup \{\lambda\}$

It has been proved that if $L$ is an even linear language then $\sigma(L)$ is regular [18]. From the last reduction, we proposed a learning algorithm in [19] that could learn even linear languages with $k$-testability based on a previous algorithm to learn $k$-testable languages in the strict sense from only positive data proposed by García *et al.* [4]. We will refer to that algorithm as `KTSS`.

In [14], we defined WKFA with local testability (in the strict sense). The basic concept is that the finite automaton obtained from the WK finite automaton by using the $ell(\cdot)$ and the $\sigma$ operations defines a regular language. So, if the obtained regular language is locally testable (in the strict sense) then, the molecule language accepted by the WK finite automaton will be locally testable (in the strict sense) in a wide sense. The class of languages accepted by $k$-testable (in the strict sense) WKFA in the upper strand will be denoted by $\mathcal{AWK}_u^{\mathcal{KLT}(\mathcal{SS})}$.

Finally, you can observe that the $\sigma$ reduction and the structural transformation $ell(\cdot)$ that we have proposed before can be easily reversed in order to obtain the initial language.

Now, we can mix up all these operations in order to learn a restricted subclass of WKFA which are still able of recognizing non-trivial context-sensitive languages. The proposed algorithm is shown as Algorithm 1.

**Algorithm 1.** An algorithm to learn $\mathcal{AWK}_u^{\mathcal{KLTSS}}$ languages from structural information

```
Input: A finite sample of linear structural duplicated strings $S$ defined over $\Sigma$
Output: A WKFA $A$ such that $S^+ \subseteq L_u(A)$
Method:
  1. $S_{ell} = ell(S)$
  2. $S_\sigma = \sigma(S_{ell})$
  3. $A_r$=KTSS($S_\sigma$)
  4. $G_{ell} = \sigma^{-1}(A_r)$
  5. $G_{lin} = ell^{-1}(G_{ell})$
  6. $A = AFWK(G_{lin})$ where $\rho = \{(a,a) : a \in \Sigma\}$
  7. Return($A$)
EndMethod.
```

The proposed algorithm is able to infer WKFA from positive structural information sample only. The restrictions over the learned WKFA are the following:

1. The linear grammar associated with the WK finite automaton generates structured strings according to $S$
2. The set $S^+$ is associated to the set $S$. Here, $S^+$ is defined by the strings obtained from $S$ by taking only the upper strand (as an application of Corollary 1).
3. The grammar obtained by reducing the corresponding linear grammar of the WK finite automaton to the regular one is $k$-testable in the strict sense. Alternatively, we can say that the WK finite automaton is $k$-testable in the strict sense as shown in [14].

4. The operation $A = AFWK(G_{lin})$ takes a linear grammar and obtains a WK finite automaton by applying the Theorem 1.

We can easily prove that there exists context-sensitive languages which are not context-free and that can be accepted in the upper strand by a WK finite automaton with the previous restrictions.

*Example 2.* Let us take the language $L = \{a^n b^n c^n : n \geq 1\}$ which can be accepted by the WK finite automaton shown in the Example 1. A positive structural information associated with the WK finite automaton could be the following

$$S = \{(a(b(c((\#)c)b)a)), (a(a(b(b(c(c(((\#)c)c)b)b)a)a)))\}$$

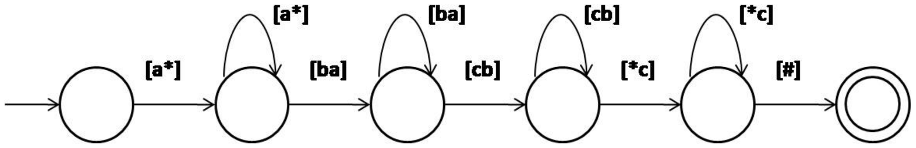
Then, by transforming the linear structures in even linear ones we obtain

$$S_{ell} = \{abc * \#cba*, aabbcc * * \#ccbbaa * *\}$$

Then, by applying the $\sigma$ transformation to $S_{ell}$ we obtain

$$S_\sigma = \{[a*][ba][cb][*c][\#], [a*][a*][ba][ba][cb][cb][*c][*c][\#]\}$$

A $k$-testable language in the strict sense inferred from the previous sample, with $k = 2$, by applying the learning algorithm KTSS, is the following one



From the last finite automaton we can obtain an even linear grammar by applying the $\sigma^{-1}$ transformation. The corresponding even linear grammar is the following

$S \to aA*$ $\quad$ $A \to aA* \mid bBa$ $\quad$ $B \to bBa \mid cCb$
$C \to cCb \mid *Dc$ $\quad$ $D \to *Dc \mid \#$

From the last even linear grammar we can obtain a linear one, by applying the homomorphism $g(a) = a$, $g(b) = b$ $g(c) = c$, $g(\#) = \#$ and $g(*) = \lambda$. The linear grammar obtained from $g$ is the following one

$S \to aA$ $\quad$ $A \to aA \mid bBa$ $\quad$ $B \to bBa \mid cCb$
$C \to cCb \mid Dc$ $\quad$ $D \to Dc \mid \#$

Observe that the WK finite automaton associated with the previous grammar is the one shown in the Example 1. In addition, the complementarity relation is obtained again from the input sample as $\rho = \{(a, a), (b, b), (c, c)\}$. So, $L \in \mathcal{AWK}_u^{\mathcal{KLTSS}}$.

The efficiency of the proposed method is shown in the following result.

**Proposition 1.** *The proposed Algorithm 1 runs in polynomial time with the size of the input sample $S$.*

*Proof.* It can be trivially proved that the Algorithm 1 runs in polynomial time. The structural transformation $ell(\cdot)$ is linear with the size of the string. The application of the $\sigma$ transformation is linear again. The application of the algorithm KTSS is polynomial time [4]. All the operations used to obtain the WK finite automaton are polynomial time given to the fact that they are the application of the $\sigma^{-1}$ and the $ell^{-1}(\cdot)$ operations. □

Finally, the identifiability of a non-trivial context-sensitive language class in the limit is shown as follows

**Proposition 2.** $\mathcal{AWK}_u^{\mathcal{KLTSS}}$ *is identifiable in the limit from only positive structural information.*

*Proof.* The identification in the limit comes from the convergence result of the algorithm KTSS exposed in [4]. So, if we apply the Algorithm 1, the identifiability in the limit of the class $\mathcal{AWK}_u^{\mathcal{KLTSS}}$ is guaranteed. □

## Generalizing the Learning Scheme

In the Algorithm 1, we have used the learning algorithm for $k$-testable languages in the strict sense, KTSS. Nevertheless, any learning algorithm for a given subclass of regular languages could be fruitful in proposing new learning algorithms for different restrictions of WKFA. So, a generalization of Algorithm 1 is proposed as Algorithm 2 which is a learning scheme to take advantages of the previously proposed reductions from WKFA to regular languages.

In the Algorithm 2, the method LearningRegPos refers to any learning algorithm that works with only positive data and obtains a finite automaton or a representation for a regular language. Observe that the learning algorithms referred

**Algorithm 2.** An algorithm to learn different families of $\mathcal{AWK}_u$ languages from structural information

```
Input: A finite sample of linear structural duplicated strings $S$ defined over $\Sigma$
Output: A WKFA $A$ such that $S^+ \subseteq L_u(A)$
Method:
  1. $S_{ell} = ell(S)$
  2. $S_\sigma = \sigma(S_{ell})$
  3. $A_r$=LearningRegPos($S_\sigma$)
  4. $G_{ell} = \sigma^{-1}(A_r)$
  5. $G_{lin} = ell^{-1}(G_{ell})$
  6. $A = AFWK(G_{lin})$ where $\rho = \{(a,a) : a \in \Sigma\}$
  7. Return($A$)
EndMethod.
```

as `LearningRegPos`, characterize different subclasses of regular languages (i.e. *k-reversible*, *terminal distinguishable*, *function distinguishable* in general, different (*positive*) *varieties* of regular languages, etc.). Furthermore, if we change the information protocol, and we provide positive structural information together with negative data as input, then we can apply other learning algorithms that work with complete data to infer regular languages.

## 4 Conclusions and Future Work

We have proposed an efficient method to infer a subclass of context-sensitive languages from linear structured positive strings. The method uses a computational device that has been previously defined in the framework of DNA computing. We think that this emerging area provides models, operations and new looks for the proposal of new learning algorithm that would enrich the map of efficiently learnable languages.

In this work, we have used structural positive information for the inference of restricted WKFA which are able of recognize non-trivial context-sensitive languages. It is an open question whether the use of different learning algorithms for some subclasses of regular languages still holds the inclusion of non-trivial context-sensitive languages in the upper strand or not. Actually, this issue is under study.

Another work that we are carrying out in the present is the application of this method to the processing of *biosequences*. Observe that in this framework the use of duplication strings comes in a natural way (i.e. DNA strings) and the availability of structural information comes from the domain task in an easy way (i.e. location of promoters, genes, motifs, etc.).

## References

1. Calera-Rubio, J., Oncina, J.: Identifying Left-Right Deterministic Linear Languages. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 283–284. Springer, Heidelberg (2004)
2. de la Higuera, C., Oncina, J.: Inferring Deterministic Linear Languages. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS (LNAI), vol. 2375, pp. 185–200. Springer, Heidelberg (2002)
3. Freund, R., Păun, G., Rozenberg, G., Salomaa, A.: Watson-Crick finite automata. In: Proceedings of DNA Based Computers III DIMACS Workshop (June 1997), pp. 297–327. The American Mathematical Society (1999)
4. García, P., Vidal, E., Oncina, J.: Learning locally testable languages in the strict sense. In: Proceedings of the First International Workshop on Algorithmic Learning Theory. Japanese Society for Artificial Intelligence, pp. 325–338 (1990)
5. Laxminarayana, J.A., Sempere, J.M., Nagaraja, G.: Learning Distinguishable Linear Grammars from Positive Data. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 279–280. Springer, Heidelberg (2004)
6. McNaughton, R., Papert, S.: Counter-free automata. MIT Press, Cambridge (1971)

7. Okawa, S., Hirose, S.: The Relations among Watson-Crick Automata and Their Relations to Context-Free Languages. IEICE Transactions on Information and Systems E89-D(10), 2591–2599 (2006)
8. Onodera, K., Yokomori, T.: Doubler and linearizer: an approach toward unified theory for molecular computing based on DNA complementarity. Natural Computing 7, 125–143 (2008)
9. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing. New computing paradigms. Springer, Heidelberg (1998)
10. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1. Springer, Heidelberg (1997)
11. Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theoretical Computer Science 76(2-3), 223–242 (1990)
12. Sakakibara, Y.: Efficient learning of context-free grammars from positive structural examples. Information and Computation 97(1), 23–60 (1992)
13. Sempere, J.M.: A representation theorem for languages accepted by Watson-Crick finite automata. Bulletin of the EATCS 83, 187–191 (2004)
14. Sempere, J.M.: On Local Testability in Watson-Crick Finite Automata. In: Vaszil, G. (ed.) Proceedings of the International Workshop on Automata for Cellular and Molecular Computing, pp. 120–128 (2007)
15. Sempere, J.M.: Exploring regular reversibility in Watson-Crick finite automata. In: Proceedings of the 13th International Symposium on Artificial Life and Robotics, pp. 505–509. AROB (2008)
16. Sempere, J.M.: Sticker expressions. In: Goel, A., Simmel, F.C., Sosik, P. (eds.) Proceedings of the 14th International Meeting on DNA Computing, pp. 200–201 (2008)
17. Sempere, J.M., Fos, A.: Learning Linear Grammars from Structural Information. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 126–133. Springer, Heidelberg (1996)
18. Sempere, J.M., García, P.: A Characterization of Even Linear Languages and its Application to the Learning Problem. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 38–44. Springer, Heidelberg (1994)
19. Sempere, J.M., García, P.: Learning Locally Testable Even Linear Languages from Positive Data. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 225–236. Springer, Heidelberg (2002)
20. Sempere, J.M., Nagaraja, G.: Learning a Subclass of Linear Languages from Positive Structural Information. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 162–174. Springer, Heidelberg (1998)
21. Takada, Y.: Grammatical inference for even linear languages based on control sets. Information Processing Letters 28(4), 193–199 (1988)
22. Zalcstein, Y.: Locally Testable Languages. Journal of Computer and System Sciences 6, 151–167 (1972)

# Polynomial Time Probabilistic Learning of a Subclass of Linear Languages with Queries

Yasuhiro Tajima and Yoshiyuki Kotani

Department of Computer and Information Sciences,
Tokyo University of Agriculture and Technology,
Naka-chou 2-24-16, Koganei, Tokyo, 184-8588, Japan
{ytajima,kotani}@cc.tuat.ac.jp

**Abstract.** We show a probabilistic learnability of a subclass of linear languages with queries. Learning via queries is an important problem in grammatical inference but the power of queries to probabilistic learnability is not clear yet. In probabilistic learning model, PAC (Probably Approximately Correct) criterion is an important one and many results have been shown in this model. Angluin has shown the ability of replacement from equivalence queries to random examples in PAC criterion but there are also many hardness results. We have shown that the class of simple deterministic languages is polynomial time learnable from membership queries and a representative sample. Also, we have shown that a representative sample can be constructed from polynomial number of random examples with the confidence probability. In this paper, we newly define a subclass of linear languages called strict deterministic linear languages and show the probabilistic learnability with membership queries in polynomial time. This learnability is derived from an exact learning algorithm for this subclass with membership queries, equivalence queries and a representative sample.

**Keywords:** learning via queries, linear language, PAC learning, representative sample.

## 1 Introduction

Learning via queries is an important problem in grammatical inference started from Angluin's work[2]. Many results about learning via queries have been shown for various language classes. A model which uses membership queries and equivalence queries is called MAT (Minimally Adequate Teacher) model and regular languages are polynomial time learnable from MAT[2]. This result is extend to some subclasses of linear languages[9].

Learning via queries and some additional information is studied about regular languages. In [1], a representative sample or a live-complete set is useful for polynomial time learning with membership queries. With this setting, a learnability of simple deterministic languages has been shown[8].

On the other hand, PAC (Probably Approximately Correct) learning model[10] is one of the most important probabilistic learning model. Learning of probabilistic deterministic finite automata[3] is studied but there are not many results about

PAC model in grammatical inference. It has been shown that equivalence query can be replaced by polynomial number of random examples in PAC criterion[2], but the power of queries to probabilistic learnability is not clear yet.

In this paper, we show a probabilistic learnability of a subclass of linear languages with membership queries. This language class called strict deterministic linear languages is newly defined and it is unambiguous and incomparable to the class of simple deterministic languages. The probabilistic learnability is derived from an exact learning algorithm for this subclass with membership queries, equivalence queries and a representative sample. Where a representative sample can be constructed from polynomial number of random example with the confidence probability. For the class of linear languages, an equivalence problem is unsolvable[5], and an equivalence problem in our newly defined languages is unknown, thus our exact learning algorithm uses powerful queries. Nevertheless, from the result of conversion from equivalence query to random examples, we can obtain a probabilistic learning algorithm of this language class.

## 2 Preliminaries

A *context-free grammar* (CFG for short) is a 4-tuple $G = (N, \Sigma, P, S)$ where $N$ is a finite set of *nonterminals*, $\Sigma$ is a finite set of *terminals*, $P$ is a finite set of *rewriting rules* (rules for short) and $S \in N$ is the *start symbol*. Let $\varepsilon$ be the word whose length is 0. If there exists no rule of the form $A \rightarrow \varepsilon$ for any $A (\neq S) \in N$, then $G$ is called $\varepsilon$-free. In this paper, we assume that every CFG is $\varepsilon$-free.

The length of $\beta$ is denoted by $|\beta|$ if $\beta$ is a string and for a set $W$, $|W|$ denotes the cardinality of $W$. For $W \subset \Sigma^*$ and $u \in \Sigma^*$, we define $u \backslash W = \{v \in \Sigma^* | uv \in W\}$ and $W/u = \{v \in \Sigma^* | vu \in W\}$.

Let $A \rightarrow \beta$ be in $P$ where $A \in N$ and $\beta \in (\Sigma \cup N)^*$. Then $\gamma A \gamma' \underset{G}{\Rightarrow} \gamma\beta\gamma'$ denotes the *derivation* from $\gamma A \gamma'$ to $\gamma\beta\gamma'$ in $G$ where $\gamma, \gamma' \in (N \cup \Sigma)^*$. We define $\overset{*}{\underset{G}{\Rightarrow}}$ to be the reflexive and transitive closure of $\underset{G}{\Rightarrow}$. When it is not necessary to specify the grammar $G$, $\alpha \Rightarrow \alpha'$ and $\alpha \overset{*}{\Rightarrow} \beta$ stand for $\alpha \underset{G}{\Rightarrow} \alpha'$ and $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$, respectively. A *word* generated from $\gamma \in (N \cup \Sigma)^*$ by $G$ is $w \in \Sigma^*$ such that $\gamma \overset{*}{\underset{G}{\Rightarrow}} w$ and the *language* generated from $\gamma$ by $G$ is denoted by $L_G(\gamma) = \{w \in \Sigma^* \mid \gamma \overset{*}{\underset{G}{\Rightarrow}} w\}$. A word generated from $S$ by $G$ for the start symbol $S$ is called a word generated by $G$ and the language generated by $G$ is denoted by $L(G) = L_G(S)$. A nonterminal $A \in N$ is said to be *reachable* if $S \overset{*}{\underset{G}{\Rightarrow}} uAw$ for some $u, w \in \Sigma^*$. and a nonterminal $D \in N$ is said to be *live* if $L_G(D) \neq \emptyset$.

A CFG $G = (N, \Sigma, P, S)$ is a *linear grammar* if every rule in $P$ is of the form $A \rightarrow uVw$ or $A \rightarrow a$ where $A \in N$ , $a \in \Sigma$, $u, w \in \Sigma^*$ and $uv \neq \varepsilon$. For every linear grammar $G$, there exists a grammar $G' = (N', \Sigma, P', S')$ such that $L(G) = L(G')$ and every rule in $G$ is of the form $A \rightarrow aBc$, $A \rightarrow aB$, $A \rightarrow Ba$ or $A \rightarrow a$ where $A, B \in N$ and $a, c \in \Sigma$.

## 3 Our Target Language

We newly define a normal form of a linear grammar. A linear grammar $G = (N, \Sigma, P, S)$ is an **RL-linear** grammar iff the followings hold.

1. Every rule in $P$ is of the form $A \rightarrow aB$, $A \rightarrow Ba$ or $A \rightarrow a$ for $A, B \in N$ and $a \in \Sigma$.
2. If $A \rightarrow aB$ and $A \rightarrow aC$ are in $P$ then $B = C$ for $A, B, C \in N$ and $a \in \Sigma$.
3. If $A \rightarrow Ba$ and $A \rightarrow Ca$ are in $P$ then $B = C$ for $A, B, C \in N$ and $a \in \Sigma$.

**Theorem 1.** *For any linear grammar $G = (N, \Sigma, P, S)$, there exists an* **RL-linear** *grammar $G' = (N', \Sigma, P', S')$ such that $L(G) = L(G')$.*

*Proof.* We suppose that every rule in $P$ is of the form $A \rightarrow aBc$, $A \rightarrow aB$, $A \rightarrow Ba$ or $A \rightarrow a$ for $A, B \in N$ and $a, c \in \Sigma$. We can make $G'$ from $G$ by the following two step conversion. Let $N' = N, P' = P, S' = S$.

1. For every rule of the form $A \rightarrow aBc$ in $P'$, replace it by $A \rightarrow aB'$ and $B' \rightarrow Bc$ where $B'$ is a new nonterminal in $N'$. After this conversion, every rule in $P'$ is of the form $A \rightarrow aB$, $A \rightarrow Ba$ or $A \rightarrow a$.
2. Delete nondeterminism in $P'$. Let $Body_R(A, a) = \{B \in N' | A \rightarrow aB \in P'\}$ and $Body_L(A, a) = \{B \in N' | A \rightarrow Ba \in P'\}$ for $A \in N'$ and $a \in \Sigma$.
   For every pair of $A \in N'$ and $a \in \Sigma$ such that $|Body_R(A, a)| \geq 2$ (or $|Body_L(A, a)| \geq 2$), delete all rules of the form $A \rightarrow aB$ (or $A \rightarrow Ba$) in $P'$, then add $A \rightarrow a \cdot Z(Body_R(A, a))$ (or $A \rightarrow Z(Body_L(A, a)) \cdot a$) where $Z(Body_R(A, a))$ (or $Z(Body_L(A, a))$) is a new nonterminal.
   In addition, for every $b \in \Sigma$ and every new nonterminal $Z(Q)(Q \subseteq N')$, add
   $$Z(Q) \rightarrow bZ(U), \quad \text{if } U \neq \emptyset,$$
   $$Z(Q) \rightarrow Z(V)b, \quad \text{if } V \neq \emptyset,$$
   and $Z(Q) \rightarrow c$ to $P'$ where
   $$U = \{C \in N | D \in Q, D \rightarrow bC \in P\},$$
   $$V = \{C \in N | D \in Q, D \rightarrow Cb \in P\},$$
   $$c \in \{d \in \Sigma | D \in Q, D \rightarrow d \in P\}.$$
   If a new nonterminal $Z(Q)$ for some $Q \subseteq N'$ newly appears then repeat this step.

Then, delete all non-reachable or non-live nonterminals in $N'$. Now, every rule $A \rightarrow \beta \in P'$, we can derive $A \overset{*}{\underset{G'}{\Rightarrow}} \beta$. It implies that $G'$ is equivalent to $G$. Moreover, $G'$ is an **RL-linear** grammar. □

We assume that every linear grammar in this paper is in **RL-linear**. It is important for grammatical inference that the grammar is unambiguous or not. If the target language is ambiguous then membership query is not powerful because it would not check the membership about nonterminals. We define the following subclass of linear grammars which is unambiguous. Learning the subclass of linear languages generated by the grammars is our goal.

**Definition 1.** *An* **RL-linear** *grammar $G = (N, \Sigma, P, S)$ is a* strict deterministic *linear grammar if the followings holds.*

- *If $A \to aB$ is in $P$ for $A, B \in N$ and $a \in \Sigma$ then every rule whose left-hand side is $A$ is of the form $A \to bC$ for some $C \in N$ and $b \in \Sigma$.*
- *If $A \to Ba$ is in $P$ for $A, B \in N$ and $a \in \Sigma$ then every rule whose left-hand side is $A$ is of the form $A \to Cb$ for some $C \in N$ and $b \in \Sigma$.*

*Thus, for every $A \in N$, there are only right linear rules or only left linear rules whose left-hand side is $A \in N$.*

For example, the following grammar is a strict deterministic linear grammar.

$$\begin{aligned} G &= (N, \Sigma, P, S) \\ N &= \{S, A, B, C, D, E\} \\ \Sigma &= \{a, b, c\} \\ P &= \{S \to aA, \\ &\quad A \to Bb, \quad A \to Cc, \quad A \to b, \quad A \to c, \\ &\quad B \to aD, \quad D \to Bb, \quad D \to b, \\ &\quad C \to aE, \quad E \to Cc, \quad E \to c \quad \} \end{aligned} \tag{1}$$

This grammar generates $a^i b^i \cup a^i c^i$ $(i \geq 1)$ which can not be generated by regular grammars or LL(1).

We denote the language class which generated by strict deterministic linear grammars by **strict-det**. A strict deterministic linear grammar $G$ satisfies that

$$S \overset{*}{\underset{G}{\Rightarrow}} uAv \iff u \backslash L(G)/v = L_G(A)$$

for any nonterminal $A$ and $u, v \in \Sigma^*$. In [7], some subclasses of deterministic linear languages and learnability are studied.

**Definition 2.** *A deterministic linear language (***DL***) is represented by a linear grammar $G = (N, \Sigma, P, S)$ such that*

- *all rules are of the form $A \to aBu$ or $A \to \varepsilon$, and*
- *if both of $A \to aBu$ and $A \to aCv$ are in $P$ then $B = C$ and $u = v$,*

*here $A, B, C \in N$, $a \in \Sigma$ and $u, v \in \Sigma^*$.*

It is shown that **DL** is identifiable in the limit from polynomial time and data[7].

**Theorem 2.** **DL** $\subset$ **strict-det**.

*Proof.* The language $a^i b^i \cup a^i c^i$ $(i \geq 1)$ which is generated by the grammar in Example (1) is **strict-det**, but it is not in **DL**[7].

Suppose that $G = (N, \Sigma, P, S)$ is a linear grammar such that $L(G)$ is in **DL**. We can make a strict deterministic linear grammar $G' = (N', \Sigma, P', S')$ from $G$ as follows.

1. Let $N' = N, P' = P, S' = S$.
2. Suppose that $A \to \varepsilon$ is in $P$. Then, delete $A \to \varepsilon$ from $P'$ and for all rules $B \to aAu$ in $P$ where $u \in \Sigma^*$, $a \in \Sigma$ and $B \in N$, add $B \to aZ_u$ to $P'$ where $Z_u$ is a new nonterminal in $N'$. Then add
$$Z_{b_1 b_2 \cdots b_n} \to Z_{b_1 b_2 \cdots b_{n-1}} b_n \quad (n = 2, 3, \cdots |u|)$$
and $Z_{b_1} \to b_1$ to $P'$ where $u = b_1 b_2 \cdots b_n$, $b_i \in \Sigma$ $(i = 1, \cdots, n)$.
3. Let $A, B \in N'$ and $a, b_i \in \Sigma$. Replace every rule $A \to aBb_1b_2 \cdots b_n$ in $P'$ with
$$A \to aZ_{Bb_1b_2\cdots b_n}$$
where $Z_{Bb_1b_2\cdots b_n}$ is a new nonterminal, then add
$$\begin{aligned} Z_{Bb_1b_2\cdots b_n} &\to Z_{Bb_1b_2\cdots b_{n-1}} b_n, \\ Z_{Bb_1b_2\cdots b_{n-1}} &\to Z_{Bb_1b_2\cdots b_{n-2}} b_{n-1}, \\ Z_{Bb_1b_2\cdots b_{n-2}} &\to Z_{Bb_1b_2\cdots b_{n-3}} b_{n-2}, \\ &\vdots \\ Z_{Bb_1} &\to Bb_1 \end{aligned}$$
to $P'$ with these new nonterminals.
4. Delete nondeterminism in $P'$ with the same method in Theorem[1].
Now, $L(G')$ is in **strict-det** and $L(G') = L(G)$.

□

## 4 Queries and Probabilistic Learning

Two types of queries are important for grammatical inference since Angluin's work[2]. Let $L_t \in$**strict-det** be the target language, and $G_t(N_t, \Sigma, P_t, S_t)$ be a strict deterministic linear grammar such that $L(G_t) = L_t$. Throughout this paper, we call the class of grammars which can generate the target language target grammars. The class of grammars by whom the learner outputs a hypothesis is called hypothesis grammars.

**Membership query.** For $w \in \Sigma^*$ as input, "yes" is responded if $w \in L_t$ and "no" is responded otherwise. $MEMBER(w) = 1$ denotes that the membership query for $w \in \Sigma^*$ responds "yes", and $MEMBER(w) = 0$ denotes that the query responds "no."

**Equivalence query.** For a hypothesis grammar $G_h$ as input, "yes" is responded if $L(G_h) = L_t$ and "no" is responded otherwise. In addition, the learner can obtain a counterexample $v \in (L_t - L(G_h)) \cup (L(G_h) - L_t)$ if the response is "no."

A learning algorithm which outputs a hypothesis $G_h$ such that $L_t = L(G_h)$ is called an exact learning algorithm. A learning algorithm which uses membership queries and equivalence queries and which outputs a hypothesis with "yes" response of an equivalence query is called an exact learning algorithm with queries.

Membership queries are useful for checking whether a word can be generated by a nonterminal or not. The result of a membership query for $uxv$ means whether $x \in L_{G_t}(A)$ or not for $A \in N_t$ if target grammars hold that

$$u \backslash L_t / v = L_{G_t}(A) \Leftrightarrow S_t \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} uwv$$

for $u, v, w \in \Sigma^*$.

We define probabilistic learning with queries as follows.

**Definition 3.** *Let $D$ be a distribution on $\Sigma^*$. The probability for $w \in \Sigma^*$ is denoted by $Pr_D(w)$ and $Pr_D(T) = \sum_{w \in T} Pr_D(w)$ for a set $T \subset \Sigma^*$. A random example is a pair of a word $w \in \Sigma^*$ drawn according to $D$ and the sign whether $w \in L_t$ or not.*

*For any distribution $D$ on $\Sigma^*$, if a learning algorithm outputs a hypothesis $G_h$ such that*

$$Pr(Pr_D((L_t - L(G_h)) \cup (L(G_h) - L_t)) \leq \varepsilon) \geq 1 - \delta$$

*for given $0 < \varepsilon \leq 1$ and $0 < \delta \leq 1$, then the learning algorithm is a probabilistic learning algorithm.*

*In addition, a probabilistic learning algorithm which uses membership queries in it is called a probabilistic learning algorithm with queries.*

We define that a language class is probabilistic learnable with queries if there exists a probabilistic learning algorithm with queries for the language class. Then, our goal is to show that **strict-det** is polynomial time probabilistic learnable with queries.

Equivalence queries in an exact learning algorithm can be replaced by polynomial number of random examples.

**Theorem 3 (Angluin(1987)).** *A learning algorithm which uses equivalence queries can be converted to a probabilistic learning algorithm without equivalence queries. In the converted algorithm, $n_i$ random examples are needed instead of $i$-th equivalence query in the original algorithm where*

$$n_i = \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + (\ln 2)(i+1)\right).$$

In other words, we can construct a wrapper algorithm which includes an exact learning algorithm with equivalence queries. When an equivalence query is asked by the included algorithm, the wrapper simulates the equivalence oracle with $n_i$ random examples. Thus, the wrapper algorithm works as a probabilistic learning algorithm from outside viewpoints.

We note that this theorem shows the one-way conversion from equivalence queries to random examples. We can not declare the exact learnability from equivalence queries and membership queries if there exists a probabilistic learning algorithm with queries.

## 5 A Representative Sample

We define a representative sample which is a set of positive examples and it covers every rule usage. A representative sample is introduced in [1] to show the polynomial time learnability of regular sets from membership queries and it. For a strict deterministic linear grammar, we define a representative sample as follows.

**Definition 4.** *A word set $R \subset \Sigma^*$ is a representative sample for a strict deterministic linear grammar $G_t$ if there exists $w \in R$ for every rule $A \to \beta$ in $P_t$ such that*

$$S_t \overset{*}{\Rightarrow} uAv \Rightarrow u\beta v \overset{*}{\Rightarrow} w$$

*where $u, v \in \Sigma^*$.*

Nevertheless, we can easily construct a representative sample for a strict deterministic linear grammar, it is difficult to determine whether a set $W \subset \Sigma^*$ is a representative sample or not for a language in **strict-det**. Thus, in our learning model, the teacher would need a strict deterministic linear grammar which generates the target language to construct a representative sample of it.

A representative sample also can be replaced by polynomial number of examples.

**Definition 5.** *Let $D$ be a distribution on $\Sigma^*$, and $G_t$ be a strict deterministic linear grammar. For a rule $A \to \beta$ in $P_t$, we define*

$$p(A \to \beta) = \sum_{S_t \underset{G_t}{\overset{*}{\Rightarrow}} uAv \Rightarrow u\beta v \underset{G_t}{\overset{*}{\Rightarrow}} w} Pr_D(w).$$

**Theorem 4 (Tajima et al.(2004)).** *Let $d = \min\{p(A \to \beta) | A \to \beta \in P_t\}$. Then, $m$ random examples contains a representative sample for $G_t$ where*

$$m > \frac{1}{d} \ln \left( \frac{|P_t|}{\delta} \right).$$

From this theorem, we can make a probabilistic learning algorithm with queries from an exact learning algorithm which uses membership queries, and a representative sample.

Combining Theorem 3 and 4, we can construct a wrapper algorithm $A_w$ such that

- $A_w$ uses random examples and membership queries,
- $A_w$ includes an exact learning algorithm which uses membership queries, equivalence queries and a representative sample, and
- $A_w$ works as a probabilistic learning algorithm with queries.

Fig.1 is the overview of the wrapper algorithm.

Teachability[4] is one of the most important study on a special examples and learnability. When we think about the teachability on grammatical inference, identification in the limit from polynomial time and data[7] is a suitable model. The

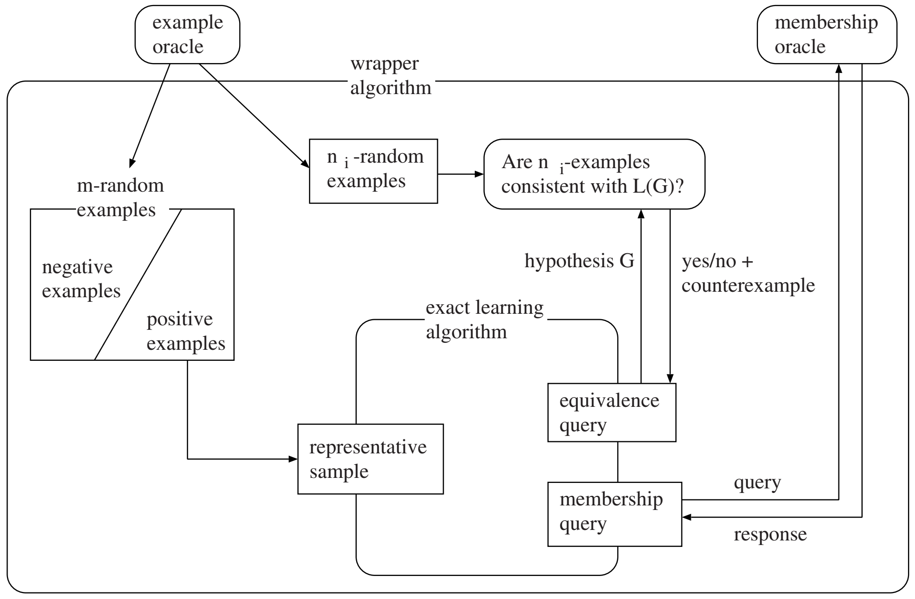


**Fig. 1.** The wrapper algorithm for probabilistic learning

learner can use a superset of special examples which helps the learning in these models.

In contrast, our learning model needs queries after a representative sample is given. Thus, we can not directly conclude that a language class which is probabilistic learnable with queries is also polynomial time teachable. The relation between our model and teachability is still unknown.

## 6 The Learning Algorithm

We show an exact learning algorithm for a strict deterministic linear language from membership queries, equivalence queries and a representative sample. It is unknown whether an equivalence problem of strict deterministic grammars is solvable or not. Nevertheless, polynomial number of random examples can substitute for both of equivalence queries and a representative sample. Thus, we can obtain a probabilistic learning algorithm with queries. This learning algorithm is a modification of our previous algorithm[8].

*[Nonterminals construction.]* If a representative sample $R$ is given, there exists a 3-tuple $(u_A, v_A, w_A) \in \Sigma^* \times \Sigma^+ \times \Sigma^*$ for every $A \in N_t$ such that $u_A v_A w_A \in R$ and $S_t \underset{G_t}{\overset{*}{\Rightarrow}} u_A A w_A \underset{G_t}{\overset{*}{\Rightarrow}} u_A v_A w_A$. Thus, the following set $M_h$ of 3-tuples is a set of candidates for nonterminals in hypothesis.

$$M_h = \{(u, v, w) \in \Sigma^* \times \Sigma^+ \times \Sigma^* | uvw \in R\}$$

We define an equivalence class on $M_h$ with a test word set $T \subset \Sigma^*$. This is a similar process of making an observation table in [2]. At the start of the learning algorithm, $T = \Sigma$. For $(u_1, v_1, w_1) \in R$ and $(u_2, v_2, w_2) \in R$, we define the equivalence relation $\overset{T}{=}$ by

$$(u_1, v_1, w_1) \overset{T}{=} (u_2, v_2, w_2) \Leftrightarrow MEMBER(u_1 x w_1) = MEMBER(u_2 x w_2)$$

for any $x \in T$.

The equivalence class for $(u, v, w) \in M_h$ is denoted by $A((u, v, w), \overset{T}{=})$ and the classification derived by $\overset{T}{=}$ on $M_h$ is denoted by $N_h = M_h / \overset{T}{=}$.

Obviously, for any $w_1, w_2 \in T$, it holds that

$$(\varepsilon, w_1, \varepsilon) \overset{T}{=} (\varepsilon, w_2, \varepsilon).$$

Thus, these $(\varepsilon, w_1, \varepsilon)$ and $(\varepsilon, w_2, \varepsilon)$ are in the same equivalence class.

*[Rules construction]* Next, candidates $Q_h$ of production rules is made. For every $(u, avb, w) \in M_h$ and $(u, a, v) \in M_h$ here $a, b \in \Sigma, u, v, w \in \Sigma^*$, we add

$$\begin{aligned} A((u, avb, w), \overset{T}{=}) &\to aA((ua, vb, w), \overset{T}{=}) \quad \text{if } (ua, vb, w) \in M_h, \\ A((u, avb, w), \overset{T}{=}) &\to A((u, av, bw), \overset{T}{=})b \quad \text{if } (u, av, bw) \in M_h, and \\ A((u, a, w), \overset{T}{=}) &\to a \end{aligned}$$

to $Q_h$.

Next, we delete inappropriate rules from $Q_h$. Let $A \to \beta$ be in $Q_h$. If there exists $t \in T$ which is an evidence such that $A$ can derive $t$ but $\beta$ can not derive it or vice versa, then $A \to \beta$ must be deleted. In other words, a rule $A \to \beta$ is inappropriate if $A \to \beta$ conflicts with results of the observation on $T$. The deletion procedure is as follows.

1. Delete $A((u, av, w), \overset{T}{=}) \to aA((ua, v, w), \overset{T}{=})$ from $Q_h$ if there exist

   $$(x_1, y_1, z_1) \in A((u, av, w), \overset{T}{=}), (x_2, y_2, z_2) \in A((ua, v, w), \overset{T}{=}) \text{and } ay \in T$$

   such that

   $$MEMBER(x_1 \cdot ay \cdot z_1) \neq MEMBER(x_2 a \cdot y \cdot z_2).$$

2. Delete $A((u, va, w), \overset{T}{=}) \to A((u, v, aw), \overset{T}{=})a$ from $Q_h$ if there exist

   $$(x_1, y_1, z_1) \in A((u, va, w), \overset{T}{=}), (x_2, y_2, z_2) \in A((u, v, aw), \overset{T}{=}) \text{and } ya \in T$$

   such that

   $$MEMBER(x_1 \cdot ya \cdot z_1) \neq MEMBER(x_2 \cdot y \cdot az_2).$$

After this deletion, if there is an inappropriate nonterminal $B \in N_h$ such that $B$ can not derive $t \in T$ but $MEMBER(utw) = 1$ where $(u, v, w) \in B$, then we delete such nonterminals at the next step. In addition, since a hypothesis should be a strict deterministic linear grammar, we must delete all rules which are of the form $A \to aB$ for $A, B \in N_h$ and $a \in \Sigma$ if there exists $b \in \Sigma$ such that $A \to bC$ is not in $Q_h$ but $MEMBER(ubxw) = 1$. These deletions can be written as follows.

3. Let $B \in N_h$ and $a \in \Sigma$. Suppose that $B \to aC$ is not in $Q_h$ for any $C \in N_h$. If there exist
$$at \in T, t \in \Sigma^* \text{and } (u, v, w) \in B$$
such that
$$MEMBER(u \cdot at \cdot w) = 1$$
then delete all rules of $B \to bD$ for every $b \in \Sigma$ and every $D \in N_h$.
4. Let $B \in N_h$ and $a \in \Sigma$. Suppose that $B \to Ca$ is not in $Q_h$ for any $C \in N_h$. If there exist
$$ta \in T, t \in \Sigma^* \text{and } (u, v, w) \in B$$
such that
$$MEMBER(u \cdot ta \cdot w) = 1$$
then delete all rules of $B \to Db$ for every $b \in \Sigma$ and every $D \in N_h$.
5. Repeat back to 3. $|R|$ times.
6. Delete all nonterminals which are not live or not reachable.

We define $S_h = A((\varepsilon, w, \varepsilon), \overset{T}{=})$.

With this process, every nonterminal $A((u, v, w), \overset{T}{=}) \in N_h$ can derive $x \in T$ such that $MEMBER(uxw) = 1$ by a linear grammar $G = (N_h, \Sigma, Q_h, S_h)$.

*[Make a hypothesis]* Now, $G = (N_h, \Sigma, Q_h, S_h)$ is a **RL-linear** grammar but is not strict deterministic linear grammar. The hypothesis $G_h$ is constructed as follows.

1. Let $P_h = \{A \to a | A \in N_h, a \in \Sigma, A \to a \in Q_h\}$.
2. Assign the type of rule to every nonterminal $A \in N_h$. If there are both form of rules such that $A \to aB$ and $A \to Bb$ where $a, b \in \Sigma$ in $Q_h$ then assign "Left" or "Right" to $A$ randomly. Otherwise, (if $A$ has only left or right linear rules,) assign the type to $A$.
3. For every $A \in N_h$ and every $a \in \Sigma$, chose a rule randomly which is of the form $A \to aB$ if $A$ is assigned "Right" or $A \to Ba$ if $A$ is assigned "Left", then add the rule to $P_h$.

Now, $G = (N_h, \Sigma, P_h, S_h)$ is a strict deterministic linear grammar and make an equivalence query for $G_h$. If a counterexample $w$ is responded then update $T$ by
$$T := T \cup \{y \in \Sigma^+ | x, z \in \Sigma^*, xyz = w\}.$$

Fig.2 is the whole learning algorithm.

```
INPUT: a representative sample $R$;
OUTPUT: correct hypothesis $G_h$;
begin
    $M_h := \{(u, v, w) \in \Sigma^* \times \Sigma^+ \times \Sigma^* | uvw \in R\}$;
    $T := \Sigma$;
    finish := 0;
    while (finish == 0)
    begin
        make $N_h$ of nonterminals with the equivalence relation $\overset{T}{=}$;
        make $Q_h$ of rules and delete inappropriate rules;
        make a hypothesis $G_h = (N_h, \Sigma, P_h, S_h)$;
        if (equivalence query for $G_h$ responds "yes")
        then
            output $G_h$, and finish := 1;
        else
            let $w \in \Sigma^*$ be the counterexample;
            $T := T \cup \{y \in \Sigma^+ | x, z \in \Sigma^*, xyz = w\}$;
        endif
    end
end.
```

**Fig. 2.** The exact learning algorithm with queries and a representative sample

Now, we show the correctness of the learning algorithm in Fig.2 and its time complexity. If the algorithm terminates then the correctness of the hypothesis is clear because of the definition of equivalence query. We are concerned with the time complexity of it.

**Lemma 1.** *The learning algorithm in Fig.2 terminates in polynomial time of* $|N_t|$*,* $|\Sigma|$*,* $|P_t|$*,* $|R|$ *and* $\max\{|w||w \in R\}$ *where* $R$ *is the given representative sample.*

*Proof.* Let $l = \max\{|w||w \in R\}$. Obviously, $|M_h| \leq \frac{l(l+1)}{2}|R|$ holds, thus $|N_h| \leq \frac{l(l+1)}{2}|R|$ also holds. For $(u, avb, w) \in M_h$, $u, v, w \in \Sigma^*$ and $a, b \in \Sigma$, there are at most 2 rules added to $Q_h$ such that

$$A((u, avb, w), \overset{T}{=}) \rightarrow aA((ua, vb, w), \overset{T}{=})$$

and

$$A((u, avb, w), \overset{T}{=}) \rightarrow A((u, av, bw), \overset{T}{=})b$$

and $A((u, a, w), \overset{T}{=}) \rightarrow a$ is added to $Q_h$ for $(u, a, w) \in M_h$. Thus, $|Q_h| < 2|M_h|$ and $|P_h| \leq |Q_h|$ hold.

Throughout the learning algorithm, $M_h$ is not increased. It implies that $|N_h|, |Q_h|, |P_h|$ are bounded by a polynomial during the learning.

Assume that a counterexample $w$ is responded by an equivalence query. Since $T$ is monotone increasing, $(u_1, v_1, w_1) \overset{T}{\neq} (u_2, v_2, w_2)$ holds once, they are never contained in the same equivalence class. It implies that $|Q_h|$ is also monotone decreasing if $N_h$ is not changed by $w$.

We can claim that if a counterexample $w$ is given, either of the following holds.

- $|Q_h|$ decreases.
- $|N_h|$ increases.

On the other hand, assume that $(u, avb, w) \in M_h$ and $(ua, vb, w) \in M_h$ correspond to a nonterminal $A \in N_t$ and $B \in N_t$ in the target grammar $G_t$, respectively, i.e. it holds that

$$S \underset{G_t}{\overset{*}{\Rightarrow}} uAw \underset{G_t}{\Rightarrow} uaBw \underset{G_t}{\overset{*}{\Rightarrow}} uavbw$$

for $(u, avb, w) \in M_h$ and $(ua, vb, w) \in M_h$. Then, the rule

$$A((u, avb, w), \overset{T}{=}) \to aA((ua, vb, w), \overset{T}{=})$$

is never deleted from $Q_h$. Thus, $Q_h$ contains at least $|P_t|$ rules. We can conclude the polynomial time termination of the learning algorithm. □

We have the main theorem.

**Theorem 5. strict-det** *is polynomial time exact learnable from membership queries, equivalence queries and a representative sample.*

*Proof.* It is clear from Lemma1 and the learning algorithm in Fig.2. □

**Theorem 6. strict-det** *is polynomial time probabilistic learnable with queries.*

*Proof.* From Theorems3 and 4, we can replace both of equivalence queries and a representative sample by polynomial number of random examples. Then, the learning algorithm becomes a probabilistic learning algorithm with queries. Obviously, its time complexity is bounded by a polynomial of $|N_t|$, $|\Sigma|$, $|P_t|$, $|R|$ and $\max\{|w||w \in R\}$. □

## 7 Conclusions

We have shown that **strict-det** is probabilistic learnable in polynomial time with membership queries. This result is derived from the polynomial time exact learning algorithm of **strict-det** from equivalence queries, membership queries and a representative sample. We can show the power of a representative sample in learning via queries but there are some problems for the future. In the study of teachability, a learning algorithm with example based queries can derive the teachability by making a teaching set contains enough many words to simulate the learning algorithm[4]. On the other hand, a representative sample can not be constructed from example based queries, thus it is hard to derive some teachability from our result. Thus, study of teachability and a representative sample is one of future works.

In [6], it has been shown that characteristic samples are important in the learning from positive and negative examples. Relation between a representative sample and characteristic sample is another future work.

## References

1. Angluin, D.: A note on the number of queries needed to identify regular languages. Info. & Contr. 51, 76–87 (1981)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Info. & Comp. 75, 87–106 (1987)
3. Clark, A., Thollard, F.: PAC-learnability of probabilistic deterministic finite state automata. J. of Machine Learning Research 5, 473–497 (2004)
4. Goldman, S.A., Mathias, H.D.: Teaching a smarter learner. J. of Comp. & Sys. Sci. 52, 255–267 (1996)
5. Harrison, M.A.: Introduction to formal language theory. Addison-Wesley, Reading (1978)
6. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27, 125–138 (1997)
7. de la Higuera, C., Oncina, J.: Inferring deterministic linear languages. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS (LNAI), vol. 2375, pp. 185–200. Springer, Heidelberg (2002)
8. Tajima, Y., Tomita, E., Wakatsuki, M., Terada, M.: Polynomial time learning of simple deterministic languages via queries and a representative sample. Theor. Comp. Sci. 329, 203–221 (2004)
9. Takada, Y.: A hierarchy of language families learnable by regular language learning. Info. & Comp. 123, 138–145 (1995)
10. Valiant, L.G.: A theory of the learnable. Comm. of the ACM 27, 1134–1142 (1984)

# How to Split Recursive Automata*

Isabelle Tellier

LIFL - Inria Lille Nord Europe
university of Lille
isabelle.tellier@univ-lille3.fr

In this paper, we interpret in terms of operations applying on extended finite state automata some algorithms that have been specified on categorial grammars to learn subclasses of context-free languages. The algorithms considered implement *specialization strategies*. This new perspective also helps to understand how it is possible to control the combinatorial explosion that specialization techniques have to face, thanks to a typing approach.

## 1 Introduction

There are often several ways to represent a language: it is well known that every regular language can be specified either by a regular grammar or by a deterministic finite state automaton. Context-free languages can also be specified by different kinds of devices. In recent previous papers [17,18], we have shown that some classes of categorial grammars (CGs in the following), generating context-free languages, could easily be represented by a family of extended automata called recursive automata (RA). This translation allowed to exhibit connexions between two previously distinct approaches of grammatical inference from positive examples: the one used in [3,13,14] to learn CGs, and the one used to learn regular grammars represented by finite state automata [1,10]. This was possible because both employ a *generalization strategy*. In particular, the generalization operators used in both contexts were shown to be similar.

Now, we want to apply the same process for *specialization strategies from positive examples*. In such strategies, the initial hypothesis is too general a grammar (or set of grammars) and each example is considered as a *constraint* which restricts the search space, until it is reduced to the target grammar. We show here that the translation of CGs into RA, which has helped to better understand the family of generalization strategies, can also help to better understand the family of specialization strategies. As a matter of fact, although barely used, specialization approaches have been proposed independently in both backgrounds: to learn subclasses of CGs in the one hand [16], and to learn regular grammars represented by finite state automata in the other hand [11]. A first move towards that direction has been briefly proposed in [19], but limited to unidirectional CGs. In this paper, we generalize the approach to its full generality.

To reach this aim, we first need to recall in section 2 how to transform a CG into a RA preserving the structures produced, in both unidirectional and

* This work was partly supported by the ANR MDCO "CroTal".

bidirectional cases. In section 3, we first briefly present the specialization strategy described by Moreau in [16], allowing to learn rigid CGs from positive examples. We then explain how it relates to the specialization strategy proposed by Fredouille and Miclet in [11], which targets regular languages represented by finite state automata. We show that Moreau's algorithm can be interpreted as some kind of well founded "state splitting" strategy applying on RA. Finaly, the whole picture is completed in section 4, by a new interpretation of yet another already known algorithm allowing to learn CGs from sentences enriched by lexical types [8,7]. It appears to be an efficiently controled specialization approach.

This paper thus proposes neither any new algorithm or result, nor any experiment, but it suggests a new stimulating look on already known strategies.

## 2 From Categorial Grammars to Recursive Automata

### 2.1 Basic Definitions of Categorial Grammars

**Definition 1 (Categories, Categorial Grammars and their Language).** *Let $\mathcal{B}$ be a set (at most countable) of basic categories containing a distinguished category $S \in \mathcal{B}$, called the axiom. $Cat(\mathcal{B})$ is the smallest set such that $\mathcal{B} \subset Cat(\mathcal{B})$ and for any $A, B \in Cat(\mathcal{B})$: $A/B \in Cat(\mathcal{B})$ and $B\backslash A \in Cat(\mathcal{B})$. Unidirectional variants allow only one of these operators (either / or \) but not both. For every finite vocabulary $\Sigma$ and for every set $\mathcal{B}$ containing $S$, a* ***categorial grammar*** *(or CG) is a finite relation $G$ over $\Sigma \times Cat(\mathcal{B})$. We note $\langle v, C\rangle \in G$ the assignment of the category $C \in Cat(\mathcal{B})$ to the element of the vocabulary $v \in \Sigma$. The syntactic rules of a CG take the form of two rewriting schemes: $\forall A, B \in Cat(\mathcal{B})$*

- *FA (Forward Application) : $A/B \quad B \rightarrow A$*
- *BA (Backward Application) : $B \quad B\backslash A \rightarrow A$*

*Unidirectional CGs make use of only one of these rules (either FA or BA) but not of both. The language generated (or recognized) by a CG $G$ is:*
*$L(G)=\{w = v_1 \ldots v_n \in \Sigma^+ \mid \forall i \in \{1,\ldots,n\}, \exists C_i \in Cat(\mathcal{B})$ such that $\langle v_i, C_i\rangle \in G$ and $C_1 \; \ldots \; C_n \rightarrow^* S\}$,*
*where $\rightarrow^*$ is the reflexive and transitive closure of the relation $\rightarrow$, defined by FA and BA schemes. For every $w \in L(G)$, a syntactic analysis structure can be produced, taking the form of a binary-branching tree whose leaf nodes are assignments of $G$ and whose internal nodes are labelled either by FA or BA and by a category (see Figure 1).*

*Example 1 (a simple CG).* CGs have mainly been used to represent natural language syntax, as illustrated by this example. Let $\mathcal{B} = \{S, T, CN\}$ where $T$ stands for "term" and $CN$ for "common noun", $\Sigma = \{John, runs, a, man, fast\}$ and $G = \{\langle John, T\rangle, \langle runs, T\backslash S\rangle, \langle man, CN\rangle, \langle a, (S/(T\backslash S))/CN\rangle, \langle fast, ((T\backslash S)\backslash(T\backslash S))\rangle\}$. This over-simple CG recognizes sentences like "John runs" or "a man runs fast" with the syntactic analysis structures of Figure 1.

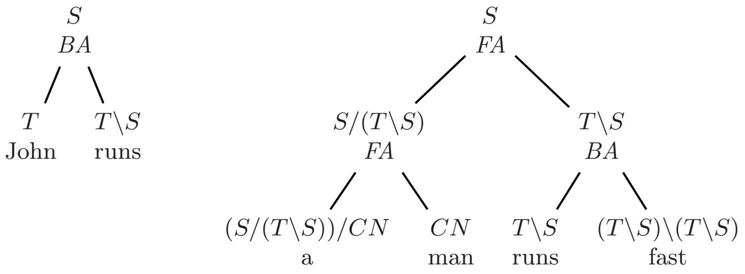


**Fig. 1.** Syntactic analysis structures produced by a CG

## 2.2 Recursive Automata and Their Language

**Definition 2 (Recursive Automaton).** *A* ***recursive automaton*** *$R$ is a* 5*-tuple $R = \langle Q, \Sigma, \gamma, q_0, F\rangle$ with $Q$ a the finite set of* states*, $\Sigma$ a finite* vocabulary*, $q_0 \in Q$ a (unique)* initial state *and $F \in Q$ a (unique)* final state*. $\gamma$ is the* transition function *of $R$, defined from $Q \times (\Sigma \cup Q)$ to $2^Q$.*

We restrict ourselves here to recursive automata (RA in the following) with unique initial and final states, but it is not a crucial choice. The only important difference between this definition and the usual definition of finite state automata is that, in a RA, it is possible to *label a transition either by an element of $\Sigma$ or by an element of $Q$*. To use a transition labelled by a state, you have to produce a string belonging to the language of this state. RA can thus be considered as special cases of "recursive transition networks" or RTRs [20]. But, depending on the notion of "state language" used, there exist in fact two distinct notions of RA which will be called, for reasons that will become clear soon, $RA_{FA}$ and $RA_{BA}$. In a $RA_{FA}$, the language $L_{FA}(q)$ associated with the state $q \in Q$ is the set of strings starting from $q$ and reaching the final state $F$, whereas in a $RA_{BA}$, $L_{BA}(q)$ is the set of strings starting from the initial state $q_0$ and reaching $q$.

**Definition 3 (Language Recognized by a RA).** *Let $R = \langle Q, \Sigma, \gamma, q_0, F\rangle$ be a $RA_{FA}$ (resp. a $RA_{BA}$). For every $q \in Q$ we define the language $L_{FA}(q)$ (resp. $L_{BA}(q)$) associated with $q$ as the smallest set satisfying:*

- *$\epsilon \in L_{FA}(F)$ (resp. $\epsilon \in L_{BA}(q_0)$);*
- *if there exists a transition labelled by $a \in \Sigma$ between $q$ and $q' \in Q$, i.e. $q' \in \gamma(q, a)$ then: $a.L_{FA}(q') \subseteq L_{FA}(q)$ (resp. $L_{BA}(q).a \subseteq L_{BA}(q')$);*
- *if there exists a transition labelled by $r \in Q$ between $q$ and $q' \in Q$, i.e. $q' \in \gamma(q, r)$ then: $L_{FA}(r).L_{FA}(q') \subseteq L_{FA}(q)$ (resp. $L_{BA}(q).L_{BA}(r) \subseteq L_{BA}(q')$).*

*The language $L_{FA}(R)$ of the $RA_{FA}$ (resp. the language $L_{BA}(R)$ of the $RA_{BA}$) is defined by: $L_{FA}(R) = L_{FA}(q_0)$ (resp. $L_{BA}(R) = L_{BA}(F)$).*

For a state $q \in Q$ such that $q \neq F$ (resp. $q \neq q_0$), the definition of $L_{FA}(q)$ (resp. of $L_{BA}(q)$) may be recursive: when it exists, it is a smallest fix-point. A real recursion occurs when, in a $RA_{FA}$, there exists a path starting from a state $q$, using a transition labelled by $q$ and reaching $F$ (resp., in a $RA_{BA}$, when there exists a path starting from $q_0$, using a transition labelled by $q$ and reaching the state $q$). Unlike finite state automata, RA are not limited to producing flat trees, because recursive transitions allow a real branching. We have shown in [19] that $RA_{FA}$ and $RA_{BA}$ are respectively linked with the two possible unidirectional CGs. This property, which justifies their name, is detailed in the following.

## 2.3 From Unidirectional CGs to RA

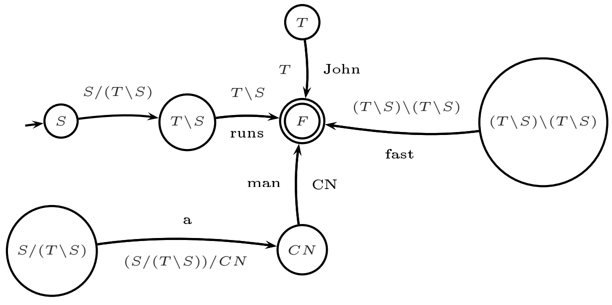
Each one of the two families of unidirectional CGs can produce any $\epsilon$-free context-free language [2]. Here, we show that every $FA$-unidirectional (resp. $BA$-unidirectional) CG can be easily transformed into a strongly equivalent $RA_{FA}$ (resp. $RA_{BA}$), i.e. generating the same structural descriptions [19]. The process, for a given $FA$-unidirectional (resp. $BA$-unidirectional) CG $G$, is the following :

– the vocabulary $\Sigma$ of the RA is the same as the one of $G$.
– let $N$ be the set of every subcategory of a category assigned to a member of the vocabulary in $G$ (a category is a subcategory of itself). The set of states for the $RA_{FA}$ (resp. $RA_{BA}$) to be built is $N \cup \{F\}$ with $F \notin N$ (resp. $N \cup \{I\}$ with $I \notin N$). The initial state is $S$ (resp. $I$), the final one is $F$ (resp. $S$).
– for every $C \in N$, define a transition labelled by $C$ between the states $C$ and $F$ (resp. between $I$ ans $C$), i.e. $F \in \gamma(C, C)$ (resp. $C \in \gamma(I, C)$).
– for every $A/B \in N$ (resp. $A\backslash B \in N$), define a transition labelled by $A/B$ (resp. $A\backslash B$) between the states $A$ and $B$, that is: $B \in \gamma(A, A/B)$ (resp. $B \in \gamma(A, A\backslash B)$).
– for every $\langle v, \rangle C \in G$, add a transition labelled by $v$ between the state $C$ and $F$, i.e. $F \in \gamma(C, v)$ (resp. add a transition between $I$ and $C$ labelled by $v$, i.e. $C \in \gamma(I, v)$).

## 2.4 Mutually Recursive Automata

Both families of unidirectional CGs have the expressivity of $\epsilon$-free context-free languages at the string level, but bidirectional CGs are useful for linguistic purposes, because of the *structures* they produce, and particularly the labels *FA* or *BA* assigned to each internal node. It is thus natural to try to extend our notion of RA to the general case of bidirectional CGs, where both $FA$ and $BA$ rules are used. As we have seen, it is possible to represent the use of $FA$ rules in a $RA_{FA}$ and the use of $BA$ rules in a $RA_{BA}$. So, we propose to represent a (bidirectional) CG by a *pair* of *mutually recursive automata* (MRA in the following): one element of the pair is a $RA_{FA}$, the other one is a $RA_{BA}$. For a syntactic analysis that uses both $FA$ and $BA$ rules, mutual calls between the two RA will be necessary. After an introducing example, we provide a general definition of MRA and give some of their properties.

*Example 2 (Example of a MRA).* Let us translate the CG $G$ given in Example 1 into a MRA (cf. Figure 2). The states of each of these RA correspond to every possible subcategory of a category assigned by $G$ to a element of the



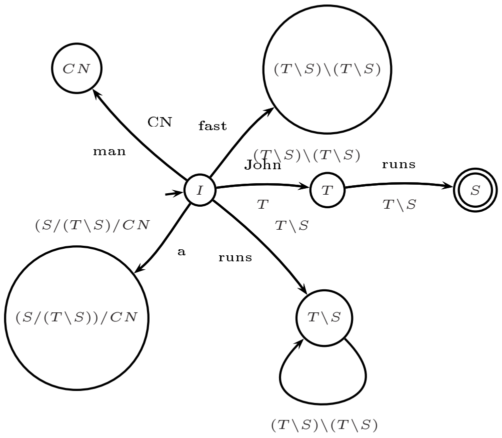


**Fig. 2.** A pair of mutually recursive automata: the $RA_{FA}$ and the $RA_{BA}$

vocabulary, plus a final state $F$ in the $RA_{FA}$ (above), and an initial state $I$ in the $RA_{BA}$ (under). The transitions have been designed exactly as explained before. Then, each RA has been simplified for readability (some un-necessary states and transitions are deleted), but not as much as possible: here, we have chosen to preserve the representation of all the final vocabulary $\Sigma$ in both automata.

**Definition 4 (MRA and their Language).** *A pair of* ***mutually recursive automata*** *(or MRA) is a pair $M = (R_{FA}, R_{BA})$ where $R_{FA} = \langle Q \cup \{F\}, \Sigma, \gamma_{FA}, S_{FA}, F\rangle$ is a $RA_{FA}$ and $R_{BA} = \langle Q \cup \{I\}, \Sigma, \gamma_{BA}, I, S_{BA}\rangle$ is a $RA_{BA}$ sharing the same vocabulary $\Sigma$ and the same set of state names $Q$ except for the final state of the $RA_{FA}$ ($F \notin Q$) and for the initial state of the $RA_{BA}$ ($I \notin Q$). We consider $\epsilon \in L_{FA}(I)$ and $\epsilon \in L_{BA}(F)$ and for every state $q \in Q$, the language $L_M(q)$ of the state $q$ in $M$ is the smallest set such that:*

- $L_{FA}(q) \cup L_{BA}(q) \subseteq L_M(q)$
- *if there exists a transition labelled by $r \in Q$ between $q$ and $q' \in Q$ in $R_{FA}$ (resp. in $R_{BA}$), i.e. $q' \in \gamma_{FA}(q, r)$ (resp. $q' \in \gamma_{BA}(q, r)$) then: $L_M(r).L_{FA}(q') \subseteq L_{FA}(q)$ (resp. $L_{BA}(q).L_M(r) \subseteq L_{BA}(q')$).*

*We define the language of the MRA as: $L(M) = L_M(S_{FA}) \cup L_M(S_{BA})$.*

For every CG $G$, there exists a MRA $M = (R_{FA}, R_{BA})$ strongly equivalent with $G$, i.e. generating the same structures.

## 3 Learning by Specialization

### 3.1 Learning Rigid CG from Positive Examples

A rigid CG is a CG in which every $v \in \Sigma$ is assigned at most one category. Kanazawa has proved [13,14] that the set of every (bidirectional) rigid CG is learnable *in the limit* (i.e. in the sense of [12]) from positive examples, i.e. from sentences. Two distinct learning algorithms are now available for this purpose. The best known is Kanazawa's, derived from "BP" (proposed earlier by Buszkowski and Penn [3]) and is a classical *generalization strategy*. The other one, called RGPL (Rigid Grammar Partial Learning) is described by Moreau in [16]. It is this second algorithm that we will concentrate on here. Although its author did not present it this way, we show that it is in fact a *specialization strategy*.

Let us first illustrate how it works on a simple example. We suppose that the available set of positive examples is {"John runs", "a man runs fast"}. At its first step, the algorithm assigns to each member of the vocabulary used at least once in the examples a distinct variable. This initial assignment is thus here:

$\mathcal{A} = \{\langle John, x_1\rangle, \langle runs, x_2\rangle, \langle a, x_3\rangle, \langle man, x_4\rangle, \langle fast, x_5\rangle\}$.

Even if a word is used several times in the examples, only one variable is introduced because the target grammar is rigid. In fact, $\mathcal{A}$ implicitely specifies *a set of grammars*: the set of rigid CGs built on the used vocabulary. As a matter of fact, every such rigid CG $G$ can be obtained by applying a substitution $\sigma$ from the set of variables to a set of categories to $\mathcal{A}$ such that:

$G = \sigma(\mathcal{A}) = \{\langle v, \sigma(C)\rangle | \langle v, C\rangle \in \mathcal{A}\}$

The substitution $\sigma$ has only the effect of renaming the variables into categories.

Of course, $\mathcal{A}$ can also be represented by a MRA $M = (R_{FA}, R_{BA})$. In this MRA, $R_{FA}$ (resp.$R_{BA}$) has $\{x_1, x_2, x_3, x_4, x_5, F\}$ (resp. $\{I, x_1, x_2, x_3, x_4, x_5\}$) as set of states, and each state $x_i$ for $1 \leq i \leq 5$ is connected to $F$ (resp. $I$ is connected to $x_i$) by a transition labelled by the corresponding word (another transition labelled by $x_i$ should be added but it is useless at this point). As $S$ appears nowhere in this MRA, the language it recognizes is empty. But it is a compact way to represent *the whole class of rigid CGs built on $\Sigma$.*

Then, each sentence is syntactically parsed with the assigments in $\mathcal{A}$, by a CYK-like algorithm. The only two possible ways to parse "John runs" are :

- either to replace $x_1$ by $S/x_2$: then a $FA$ rule can be applied
- either to replace $x_2$ by $x_1\backslash S$: then a $BA$ rule can be applied

These kinds of substitutions express a *constraint* that the variables ($x_1$ or $x_2$) must satisfy: $\mathcal{A}$ must thus be updated to a disjunction of sets of assignments, each subset corresponding to a subclass of rigid CGs. A simpler way to store the current subsets of possible solutions is to store the set of possible substitutions that can be applied to $\mathcal{A}$. In our case, this set is made of $\{\sigma_1, \sigma_2\}$, with $\sigma_1(x_1) = S/x_2$ and is equal to the identical function elsewhere, and $\sigma_2(x_2) = x_1\backslash S$ and is equal to the identical function elsewhere. $\sigma_1(\mathcal{A})$, as well as $\sigma_2(\mathcal{A})$, can be represented by a MRA derived from the previous one. This time, both MRA recognize exactly the sentence "John runs".

To parse "a man runs fast", many more solutions are possible. The maximum theoretical number is $5 * 2^3 = 40$ because there are 5 possible binary branching trees with 4 leaves (this can be computed in the general case by the Catalan number), and each of them has 3 internal nodes which can receive either a $FA$ or a $BA$ label. This makes $40 * 2 = 80$ theoretical possible substitutions by combining the constraints obtained from both sentences (the combinaison is a classical composition of functions). But, among them, some are contradictory: as the target grammar is rigid, the unique category assigned to the word "runs" cannot be of the form $x_i/x_j$ and $x_k\backslash x_l$ at the same time. We thus see where the initial class plays a role in the learning strategy.

It is easy to see that the main problem with this algorithm is the combinatorial explosion it has to face, especially when examples do not share any common word. This is not surprising, since the problem of learning rigid CGs from sentences is known to be NP-hard [4].To limit this explosion, Moreau proposes to exploit as much initial knowledge as possible, in the form of an *initial grammar*, that is, initially known categories for some usual words (for example the lexical ones) which cannot be renamed, as it is the case for the variable categories.

Furthermore, there is no guarantee at all that this strategy always converges to a unique solution. In theory, to fulfill the requirements of learnabiblity *in the limit*, when several possible compatible grammars are available, inclusion tests should be performed to select the one generating the "smallest" language. This problem also occurred with Kanazawa's algorithm, when applied to sentences.

### 3.2 State Merges and State Splits

The previous strategy can now be interpreted in terms of operations applying on MRA. As we have seen, at every step of this algorithm, the search space is a disjunction of sets of assignments of the form $\sigma(\mathcal{A})$ for some substitution $\sigma$, and each of them can be represented by a MRA. The MRA corresponding with $\mathcal{A}$ recognizes no sentence. But, as soon as at least one example has been treated (and, thus, the category $S$ been introduced), $\sigma(\mathcal{A})$ specifies a set of CGs recognizing at least this example. What is the effect of a constraint on a MRA ?

The constraints always take the form: $x_k = x_l$, where $x_k$ and $x_l$ are already introduced variables or equal to $S$, or $x_k = X_m/X_n$ or $x_k = X_m\backslash X_n$ , with $X_m$ and $X_n$ any category built on the set of every variable union S.

– the effect of a constraint of the form $x_k = x_l$ on a MRA is a *state merge* in both the $RA_{FA}$ and the $RA_{BA}$ of the MRA. As, in MRA, $x_k$ can also be used as transition labels, corresponding *transition merges* can also occur.
– the effect of a constraint of the form $x_k = X_m/X_n$ (resp. $X_m\backslash X_n$) can be decomposed into four steps:

1. $X_m/X_n$ (resp. $X_m\backslash X_n$) replaces $x_k$ everywhere in the MRA;
2. every subcategory of $X_m$ and $X_n$ (including themselves) not already identified (i.e. not already a sub-category of the previous set of assignments) becomes a new state in both RA: in the $RA_{FA}$, it is linked to the state $F$ (resp. in the $RA_{BA}$ from the state $I$) by a transition labelled by its name;
3. in the $RA_{FA}$ (resp. in the $RA_{BA}$), a new transition labelled by $X_m/X_n$ (resp. $X_m\backslash X_n$) links the states $X_m$ and $X_n$, and the same occurs for every newly identified subcategory;
4. the states and transitions of the same name are merged in each RA.

This operation can now be compared to the "state splitting strategy" proposed by Fredouille and Miclet in [11] to learn regular languages represented by finite state automata by specialization. For example, the constraint $x_1 = S/x_2$ has the effect of splitting the state $x_1$ into two new states: $S$ and $x_2$. Then, as a state named $x_2$ already exists, the new one is merged with the previous one. But our specialization operation is more general than Fredouille and Miclet's, because of the recursive nature of the automata on which it applies. Furthermore, their algorithm was a specialization strategy *at the language level*: their initial hypothesis was the most general regular language $\Sigma^*$ and constraints were used to *specialize the language*. Moreau's algorithm is a specialization strategy *at the set of grammars level*: its initial hypothesis is the set of possible grammars, and the examples are used to introduce constraints that reduce this set to subsets. The corresponding MRA represents a *set of grammars* and not only a specific language. For these reasons, our approach cannot be easily adapted to usual finite state automata. But we believe that our state splitting operator is better founded than the previous one, because it is the formal counterpart of well-defined substitutions.

## 4 Learning from Typed Examples Revisited

We now show that the algorithm proposed in [8,7] to learn CGs from typed examples can be considered as a specialization strategy where state splits and state merges are controlled by a typing approach.

### 4.1 Learning from Semantically Typed Examples

The idea of learning CGs from typed examples was first introduced in [8]. The types considered in this work are borrowed from Montague's theory [6]: they are lexicalized terms derived from syntactic categories by a morphism, and coincide with the type of the logical formula that translates the associated word. Learning from typed examples is cognitively relevant because types can be interpreted as semantic information available in the environment or previously learned. In this section, we briefly recall this notion in a general fashion and give the conditions under which they can help learning.

The notion of types useful for learning CGs is based on:

- a finite set $\tau$ of basic types among which is a distinguished type $t \in \tau$ standing for "truth values": usually, this set is $\tau = \{e, t\}$ where $e \in \tau$ is the type of "entities". Montague also used a type $s$ for "intensions" that will not be used in the following;
- the set $\mathit{Types}(\tau)$ of every type is the smallest set such that $\tau \subset \mathit{Types}(\tau)$ and for every type $\alpha, \beta \in \mathit{Types}(\tau)$, $\langle \alpha, \beta \rangle \in \mathit{Types}(\tau)$. $\langle \alpha, \beta \rangle$ is the type of functions that require an argument of type $\alpha$ and provide a result of type $\beta$.

Types in $\mathit{Types}(\tau)$ are useful for learning a CG only if they are connected with its syntactic categories in $\mathit{Cat}(\mathcal{B})$. More precisely, the necessary condition to be fulfilled is that there exists a homomorphism $h$ such that:

- for every basic category $C \in \mathcal{B}$, $h(C)$ is defined and belongs to $\mathit{Types}(\tau)$. The distinguished category $S \in \mathcal{B}$ is associated with the distinguished type $t \in \tau$: $h(S) = t$.
- for every other category in $\mathit{Cat}(\mathcal{B})$ of the form $A/B$ or $A\backslash B$, we have: $h(A/B) = h(B\backslash A) = \langle h(B), h(A) \rangle$.

*Example 3 (classical semantic types for natural languages).* Let $\tau = \{e, t\}$. The words of the grammar defined in Example 1 receive the following semantic types:

- "John" can be considered as an entity of type $e$;
- "runs" and "man" are one-place predicates; their type is: $\langle e, t \rangle$;
- "fast" is a "one-place-predicate modifier", i.e. it transforms a predicate of arity one into another one of the same arity: it thus has the type $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$;
- finaly, if we follow Montague's intuition about the "proper treatment of quantification" [6], the determiner "a" has the most complex type: $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$.

The corresponding homomorphism $h$ is defined by: $h(S) = t$, $h(T) = e$, $h(CN) = \langle e, t \rangle$. As required, if $\langle v, C \rangle \in G$, the semantic type of $v$ is $h(C)$.

## 4.2 How Types Help to Control State Splits and State Merges

Learning from typed examples means learning from sentences where each element of the vocabulary $v \in \Sigma$, which should be assigned $C \in \mathit{Cat}(\mathcal{B})$ by the target grammar $G$ to analyse this sentence, is provided with the corresponding type $h(C) \in \mathit{Types}(\tau)$. As we will see, the learning strategy proposed in [8,7] can also be interpreted in terms of operations applying on MRA. We illustrate this algorithm on our example. The input data are now of the form:

| John | runs |
|---|---|
| $e$ | $\langle e, t \rangle$ |
| $e$ | $x_1 \langle e, t \rangle$ |

| a | man | runs | fast |
|---|---|---|---|
| $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$ | $\langle e, t \rangle$ | $\langle e, t \rangle$ | $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$ |
| $x_2 \langle x_3 \langle e, t \rangle, x_4 \langle x_5 \langle e, t \rangle, t \rangle \rangle$ | $x_6 \langle e, t \rangle$ | $x_1 \langle e, t \rangle$ | $x_7 \langle x_8 \langle e, t \rangle, x_9 \langle e, t \rangle \rangle$ |

In these typed examples, the third line is the result of a simple pre-treatment which consists in introducing variables in front of every "functional type". The variables are all distinct, except when the same couple "word, type" occurs (as it is the case here for the couple "runs, $\langle e, t \rangle$"). These variables will eventually take the value "/" or "\" during the learning process. The initial set of assignements is, this time:

$\mathcal{A} = \{\langle John, e\rangle, \langle runs, x_1\langle e,t\rangle\rangle, \langle a, x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle, t\rangle\rangle\rangle, \langle man, x_6\langle e,t\rangle\rangle,$
$\langle fast, x_7\langle x_8\langle e,t\rangle, x_9\langle e,t\rangle\rangle\rangle\}.$

As previously, $\mathcal{A}$ implicitely specifies a *set of grammars*. This set is much larger than the one of rigid CGs: it is the set of CGs which can assign an arbitrary number of distinct categories to each word (so, it intersects every class of $k$-valued CGs), but for which there exists a homomorphism such that every distinct category assigned to the same word gives rise to a distinct type. In formal terms, it is such that there exists a homomorphism $h$ satisfying:

$\forall \langle v, C_1\rangle, \langle v, C_2\rangle \in G,\ h(C_1) = h(C_2) \Longrightarrow C_1 = C_2.$

We have shown [9] that for every $\epsilon$-free context-free language, it is possible to define a CG generating this language, a set of types and a homomorphism such that this property is satisfied. This new target class is learnable in the limit from typed examples [8,7].

As previously, $\mathcal{A}$ can also be represented by a MRA. But the information carried by the types is much richer than the one carried by the basic variables Moreau used: types can be interpreted as some kind of maximal bound on the possible categories they replace; they display all their potential renaming.

The learning algorithm applies as in section 3.1: it consists in trying to parse each sentence with the rules *FA* and *BA* adapted to types so as to reach the type $t$ at the root, by defining constraints on the variables (see [7] for details). The only possible type-compatible way to parse the first typed example "John runs" is to have: $x_1 = \backslash$, meaning that only a *BA* rule is compatible with the type assignments. "runs" should thus finaly receive the category $e\backslash t$. This time, there is only one type-compatible way to parse "a man runs fast": this parse (isomorphic to the one in Figure 1) is shown on Figure 3. Both typed examples lead to the following (unique) set of constraints: $x_2 = /$, $x_3 = x_6$, $x_7 = \backslash$, $x_8 = x_1 = \backslash$, $x_4 = /$, $x_5 = x_9$.

The set of assignments is thus updated to:

$\mathcal{A} = \{\langle John, e\rangle, \langle runs, \backslash\langle e,t\rangle\rangle, \langle a, /\langle x_3\langle e,t\rangle, /\langle x_5\langle e,t\rangle, t\rangle\rangle\rangle, \langle man, x_3\langle e,t\rangle\rangle,$
$\langle fast, \backslash\langle\backslash\langle e,t\rangle, x_5\langle e,t\rangle\rangle\rangle\}.$

If we apply the process of section 2.4 to this set (after re-ordering the types to make them similar to syntactic categories and $t$ playing the role of $S$), we obtain the MRA of Figure 4. In this example, with only two typed examples, we obtain a unique MRA which is nearly isomorphic to the target one.

In this context, the constraints take the form $x_i = x_j$, $x_i = /$ or $x_i = \backslash$ and give rise to the same transformations as the one detailed in section 3.2. It could seem that the first kind corresponds to a state merge and the other two to a

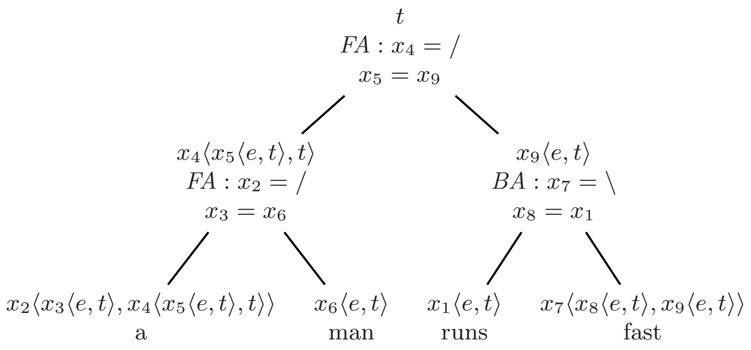
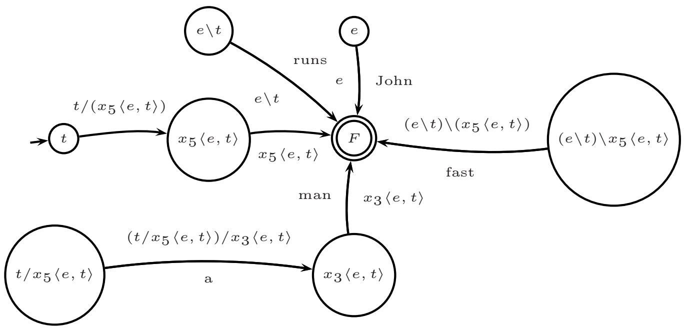

$t$
$FA : x_4 = /$
$x_5 = x_9$
$x_4\langle x_5\langle e,t\rangle, t\rangle$
$FA : x_2 = /$
$x_3 = x_6$
$x_9\langle e,t\rangle$
$BA : x_7 = \backslash$
$x_8 = x_1$
$x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle, t\rangle\rangle$
a
$x_6\langle e,t\rangle$
man
$x_1\langle e,t\rangle$
runs
$x_7\langle x_8\langle e,t\rangle, x_9\langle e,t\rangle\rangle$
fast

**Fig. 3.** Parse tree for a typed example


$e\backslash t$
$e$
runs
$e$
John
$t/(x_5\langle e,t\rangle)$
$e\backslash t$
$t$
$x_5\langle e,t\rangle$
$F$
$x_5\langle e,t\rangle$
$(e\backslash t)\backslash(x_5\langle e,t\rangle)$
$(e\backslash t)\backslash x_5\langle e,t\rangle$
fast
man
$x_3\langle e,t\rangle$
$(t/x_5\langle e,t\rangle)/x_3\langle e,t\rangle$
$t/x_5\langle e,t\rangle$
a
$x_3\langle e,t\rangle$

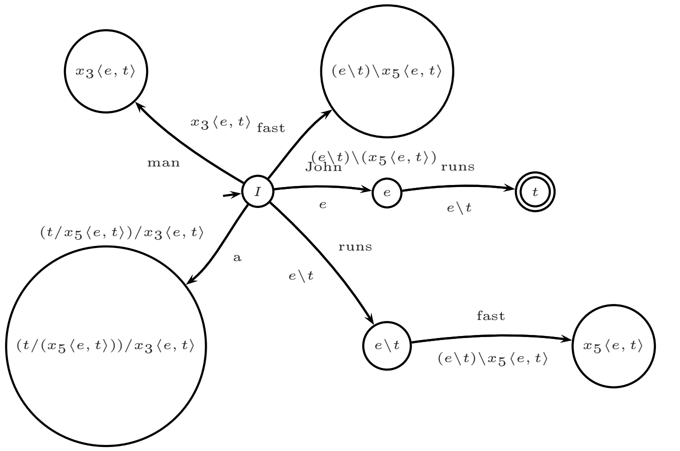

$x_3\langle e,t\rangle$
$(e\backslash t)\backslash x_5\langle e,t\rangle$
$x_3\langle e,t\rangle$
fast
man
$(e\backslash t)\backslash(x_5\langle e,t\rangle)$
John
runs
$I$
$e$
$t$
$e$
$e\backslash t$
$(t/x_5\langle e,t\rangle)/x_3\langle e,t\rangle$
a
runs
$e\backslash t$
fast
$(t/(x_5\langle e,t\rangle))/x_3\langle e,t\rangle$
$e\backslash t$
$(e\backslash t)\backslash x_5\langle e,t\rangle$
$x_5\langle e,t\rangle$

**Fig. 4.** MRA for type assignments

state split, but the situation is a bit more complex. In our example, to reach the target, only one constraint is missing: $x_5 = \backslash$. The typed example corresponding to the sentence "John runs fast" would provide this constraint. Its first effect on

| *vocabulary* | *Moreau's initial assigment* | *target category* | *pre-treated initial assignment types* |
|---|---|---|---|
| *John* | $x_1$ | $T$ | $e$ |
| *a* | $x_2$ | $(S/(T\backslash S)/CN$ | $x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle, t\rangle\rangle$ |
| *man* | $x_3$ | $CN$ | $x_6\langle e,t\rangle$ |
| *runs* | $x_4$ | $T\backslash S$ | $x_1\langle e,t\rangle$ |
| *fast* | $x_5$ | $(T\backslash S)\backslash(T\backslash S)$ | $x_7\langle x_8\langle e,t\rangle, x_9\langle e,t\rangle\rangle$ |

**Fig. 5.** Tabular showing the starting points and target of the two algorithms

the MRA would be to rename the state $x_5\langle e,t\rangle$ both in the $RA_{FA}$ and in the $RA_{BA}$ by $\backslash\langle e,t\rangle$, that is $e\backslash t$. But, doing so, this state becomes identical to an already existing one and then must be merged to it.

The table of Figure 5 explains why types help the algorithm to avoid a combinatorial explosion and to converge quicker. We have seen that there always exists a homomorphism $\sigma$ between column 2 and column 3, which is the target of the learning process. Hypotheses about types ensure that there also exists a homomorphism $h$ between column 3 and column 4. This situation is similar to the one described in [15], and analyzed in [5]. The two learning algorithms presented here are both specialization strategies at the *set of grammars level*, but their initial hypothesis is either a lower bound or an upper bound of the set of categories of the target grammar. Types are efficient because they allow to control the possible renamings.

## 5 Conclusion

In grammatical inference from positive examples, two sources of information are usually available: the target class and the set of examples. Generalization techniques use the examples to generate a "most specific grammar" compatible with them (the prefix tree automaton in the case of regular languages), and then use the target class to generalize it. Specialization techniques do the contrary: the class is the starting point and the examples help to specialize it.

In this paper, we propose a new perspective on these techniques. First, we see that disjunctions of MRA are able to represent the *search space* of such learning algorithms. Second, we show that the algorithm to learn CGs from typed examples proposed in [8,7] introduces type control into the process. The initial semantic types associated with the elements of the vocabulary specify some kind of maximal bound on the possible renamings, allowing to limit the combinatorial explosion of solutions.

## References

1. Angluin, D.: Inference of Reversible Languages. Journal of the ACM 3, 741–765 (1982)
2. Bar Hillel, Y., Gaifman, C., Shamir, E.: On Categorial and Phrase Structure Grammars. Bulletin of the Research Council of Israel 9F (1960)

3. Buszkowki, W., Penn, G.: Categorial grammars determined from linguistic data by unification, newblock. Studia Logica, 431–454 (1990)
4. Costa-Florencio, C.: Consistent Identification in the Limit of Rigid Grammars from Strings Is NP-hard. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 49–62. Springer, Heidelberg (2002)
5. Coste, F., Fredouille, D., Kermovant, C., de la Higuera, C.: Introducing domain and typing bias in automata inference. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 115–126. Springer, Heidelberg (2004)
6. Dowty, D., Wall, R.E., Peters, S.: Introduction to Montague Semantics. Linguistics and Philosophy, Reidel (1981)
7. Dudau-Sofronie, D.: Apprentissage de grammaires catégorielles pour simuler l'acquisition du langage naturel á l'aide d'informations sémantiques, thése d'informatique, université Lille3 (2004)
8. Dudau-Sofronie, D., Tellier, I., Tommasi, M.: Learning categorial grammars from semantic types. In: Proceedings of the 13th Amsterdam Colloquium, pp. 79–84 (2001)
9. Dudau-Sofronie, D., Tellier, I., Tommasi, M.: A Learnable Class of CCGs from Typed Examples. In: Proceedings of the 8th conference on Formal Grammars, pp. 77–88 (2003)
10. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 25–37. Springer, Heidelberg (1994)
11. Fredouille, D., Miclet, L.: Experiences sur l'inference de langage par specialisation. In: Proceedings of CAP 2000, pp. 117–130 (2000)
12. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
13. Kanazawa, M.: Identification in the limit of categorial grammars. Journal of Logic, Language and Information 5(2), 115–155 (1996)
14. Kanazawa, M.: Learnable Classes of Categorial Grammars. CSLI Publications (1998)
15. Kermovant, C., de la Higuera, C.: Learning language with help. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 161–173. Springer, Heidelberg (2002)
16. Moreau, E.: Apprentissage partiel de grammaires lexicalisées. TAL 45(3), 71–102 (2004)
17. Tellier, I.: When categorial grammars meet regular grammatical inference. In: Blache, P., Stabler, E.P., Busquets, J.V., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 301–316. Springer, Heidelberg (2005)
18. Tellier, I.: Learning recursive automata from positive examples. In: Revue d'Intelligence Artificielle. New Methods in Machine Learning(20/2006), pp. 775–804 (2006)
19. Tellier, I.: Grammatical inference by specialization as a state splitting strategy. In: Proceedings of the 16th Amsterdam Colloquium, pp. 223–228 (2007)
20. Woods, W.A.: Transition Network Grammars of Natural Language Analysis. Communication of the ACM 13, 591–606 (1970)

# A Note on the Relationship between Different Types of Correction Queries[⋆]

Cristina Tîrnăucă

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
cristina.bibire@estudiants.urv.es
http://www.grlmc.com

**Abstract.** The adult-child interaction which takes place during the child's language acquisition process has been the inspiration for Angluin's teacher-learner model [1], the forerunner of today's active learning field. But the initial types of queries have some drawbacks: equivalence queries are both unrealistic and computationally costly; membership queries, on the other hand, are not informative enough, not being able to capture the feedback received by the child when he or she makes mistakes. This is why a new type of query (called correction query), weaker than the first one and more informative than the second, appeared. While in the case of natural languages it is well understood what correcting means, in formal language theory different objects may require different types of corrections. Therefore, several types of correction queries have been introduced so far. In this paper we investigate the relations existing between different models of correction queries, as well as their connection to other well-known Gold-style and query learning models. The study comprises results obtained in the general case when time complexity issues are ignored, and in the restricted case when efficiency constraints are imposed.

**Keywords:** query learning, Gold-style learning, correction query.

## 1 Introduction

The way children learn their mother language is an amazing process. They receive examples of words in the vocabulary and sentences in that language, and after some transitory period - in which they still make mistakes and are corrected by adults - they are suddenly able to express themselves fluently and errorless.

It has been argued that the formal model that best describes the child-adult interaction within the process of child acquiring his or her native language is the *query learning model* [1]. The most investigated types of queries, and in the

[⋆] This work was possible thanks to the FPU Fellowships AP2004-6968 from the Spanish Ministry of Education and Science.

same time the first introduced, are *membership queries* (MQs) and *equivalence queries* (EQs).

There are quite a few reasons though for which people working in grammatical inference, and especially in active learning, have been trying to find effective query learning algorithms that avoid the use of EQs. First of all, EQs are computationally costly. Secondly, they are quite unnatural for a real-life setting: no child would ever ask his mother if the current hypothesis represents the correct grammar of the language. Finally, it might happen that the teacher does not even have a grammar for the target language - take, for example, the case of native speakers that did not ever studied grammar.

On the other hand, when answering a MQ in the negative way, no other information is provided by the teacher. Inspired by the way adults guide the process of children's language acquisition by correcting them when necessary, the authors of [2] propose a modified version of MQs, called *correction query* (CQ), that incorporates this idea. More precisely, the difference consists in the fact that for strings not belonging to the target language, the teacher must provide the learner with a *correction*.

Whereas in a real-life setting correcting an ungrammatical utterance is done, most of the times, by replacing the error with a correct (sequence of) word(s), if the target is a formal language, then one needs to adapt the type of correction to best suit the particularities of that language class. And indeed, since their introduction, several types of corrections have been proposed in order to learn different objects: prefix correction queries [2] and length bounded correction queries [3] for deterministic finite automata, edit distance based correction queries for balls of strings [4], regular expressions and pattern languages [5], the nearest positive example [6] for sets of positive integers, and structural correction queries for regular tree languages [7].

Intuitively, this new type of query can be placed somewhere between MQs and EQs. But what exactly can we learn with CQs, and what can be done in polynomial time? These are the questions whose answers constitute the object of the present paper. The first steps in this research direction have been done already: Tîrnăucă and Kobayashi [8] investigate the relations existing between the model of learning with prefix correction queries (PCQs) and other well-known Gold-style and query learning models when time complexity issues are neglected (they show, among other things, that learning with PCQs is strictly less powerful than learning with EQs, and more powerful than the model of learning with MQs). Moreover, this study is continued in [9] by imposing efficiency constraints.

In this paper we focus on the identification of formal languages ranging over indexable classes of non-empty recursive languages as target concepts when the learner is allowed to ask one of the other two types of CQs: length bounded correction queries (LBCQs) and edit distance correction queries (EDCQs).

The article is organized as follows. Preliminary notions are presented in Section 2. In the third section we show that when we neglect time complexity issues, learning with LBCQs or EDCQs is basically the same as learning with MQs. In Section 4 we concentrate on polynomial time algorithms and we present the relations between different models of learning with CQs. We conclude with further remarks and future work topics in Section 5.

## 2 Preliminaries

Let $\Sigma$ be a finite set of symbols called *alphabet*, and $\Sigma^*$ the set of strings over $\Sigma$. A *language* $L$ over $\Sigma$ is a subset of $\Sigma^*$. The elements of $L$ are called *strings*. Let $u$, $v$, $w$ be strings in $\Sigma^*$ and $|w|$ be the length of the string $w$. $\lambda$ is a special string called the *empty string* and has length 0. We denote by $uv$ the concatenation of the strings $u$ and $v$. If $w = uv$ for some $u, v$ in $\Sigma^*$, then $u$ is a *prefix* of $w$ and $v$ is a *suffix* of $w$. By $Tail_L(u)$ we denote the set $\{v \mid uv \in L\}$.

Assume that $\Sigma$ is a totally ordered set, and let $\prec_L$ be the lexicographical order on $\Sigma^*$. Then, the *lex-length order* $\prec$ on $\Sigma^*$ is defined by: $u \prec v$ if either $|u| < |v|$, or else $|u| = |v|$ and $u \prec_L v$. In other words, strings are compared first according to length and then lexicographically. In the rest of the paper strings are always compared with respect to the lex-length order.

The edit distance between two strings $w$ and $w'$, denoted $d(w, w')$ in the sequel, is the minimum number of *edit operations* needed to transform $w$ into $w'$ and can be computed in $O(|w| \cdot |w'|)$ time by dynamic programming [10]. The edit operations are either (1) *deletion*: $w = uav$ and $w' = uv$, or (2) *insertion*: $w = uv$ and $w' = uav$, or (3) *substitution*: $w = uav$ and $w' = ubv$, where $u, v \in \Sigma^*, a, b \in \Sigma$ and $a \neq b$.

### 2.1 Learning Models

An *indexed family* (or *indexable class*) Let $\mathcal{C} = (L_i)_{i \geq 1}$ is a recursive enumeration of non-empty languages such that membership in $L_i$ is uniformly decidable for all $i \geq 1$, i.e., there is a computable function that, for any $w \in \Sigma^*$ and $i \geq 1$, returns 1 if $w \in L_i$, and 0 otherwise.

**Gold-Style Learning.** An *inductive inference machine* (IIM) $\mathcal{A}lg$ is an algorithmic device that reads longer and longer initial segments $\sigma$ of a text (informant), and outputs numbers as its hypotheses. Given a text (an informant) $\sigma$ for a language $L \in \mathcal{C}$, $\mathcal{A}lg$ *learns* $L$ *from* $\sigma$ if the sequence of hypotheses output by $\mathcal{A}lg$, when fed $\sigma$, stabilizes on a number $i$ with $L_i = L$. We say that $\mathcal{A}lg$ *learns* $\mathcal{C}$ *from text (informant)* if it identifies each $L \in \mathcal{C}$ from every corresponding text (informant). A slightly modified version of the learning in the limit model is the so-called model of *conservative learning* (see [12,13] for more details). A conservative IIM is only allowed to change its mind in case its actual guess contradicts the data seen so far.

We denote by *LimTxt* (*LimInf*) the collection of all indexable classes $\mathcal{C}$ for which there is an IIM $\mathcal{A}lg$ such that $\mathcal{A}lg$ identifies $\mathcal{C}$ from text (informant). One can similarly define *ConsvTxt* and *ConsvInf* for which the inference machines should be conservative IIMs.

Although an IIM is allowed to change its mind finitely many times before returning its final and correct hypothesis, in general it is not decidable whether or not it has already output its final hypothesis. In case that for a given indexable class $\mathcal{C}$, there exists an IIM $\mathcal{A}lg$ such that given any language $L \in \mathcal{C}$ and any text (or informant) for $L$, the first hypothesis $i$ output by $\mathcal{A}lg$ is already correct (i.e., $L_i = L$), we say that $\mathcal{A}lg$ *finitely identifies* $\mathcal{C}$ (see [14]). The corresponding models *FinTxt* and *FinInf* are defined as above.

**Query Learning.** In this model a learner has access to an oracle or a teacher that truthfully answers queries of a specified kind. Conform [15], a *query learner* $\mathcal{A}lg$ is an algorithmic device that, depending on the reply of the previous queries, either computes a new query, or returns a hypothesis and halts. More formally, let $\mathcal{C}$ be an indexable class and $L$ an arbitrary language in $\mathcal{C}$. The query learner $\mathcal{A}lg$ *learns* $L$ *using some type of queries* if it eventually halts and its only hypothesis, say $i$, correctly describes $L$ (i.e., $L_i = L$). So, $\mathcal{A}lg$ returns its unique and correct guess $i$ after only finitely many queries. Moreover, $\mathcal{A}lg$ *learns the class* $\mathcal{C}$ *using some type of queries* if it learns every language of that class using queries of the specified type.

Apart from the well-known membership and equivalence queries, in this paper we investigate three types of correction queries. Let $L$ be a language over the alphabet $\Sigma$ and $w$ a string in $\Sigma^*$.

– **Prefix correction queries** (Becerra, Dediu, Tîrnăucă [2])
  The *prefix correcting string of* $w$ *with respect to* $L$, denoted $C_L(w)$, is

$$C_L(w) = \begin{cases} \min\{v \mid v \in \mathit{Tail}_L(w)\}, & \text{if } \mathit{Tail}_L(w) \neq \emptyset \\ \Theta, & \text{otherwise.} \end{cases}$$

  Hence, $C_L$ is a function from $\Sigma^*$ to $\Sigma^* \cup \{\Theta\}$. Note that $C_L(w)$ is $\lambda$ if and only if $w \in L$, and $C_L(w)$ equals $\Theta$ if and only if $w$ is not the prefix of any of the strings in $L$.
– **Length bounded correction queries** (Tîrnăucă [3])
  Let us fix an integer $l$. The $l$*-bounded correction of* $w$ *with respect to* $L$, denoted $C_L^l(w)$, is

$$C_L^l(w) = \{v \in \mathit{Tail}_L(w) \mid |v| \leq l\}.$$

  So, $C_L^l$ is a function from $\Sigma^*$ to $\mathcal{P}(\Sigma^*)$. Note that $\lambda \in C_L^l(w)$ if and only if $w \in L$.
– **Edit distance correction queries** (Becerra et al. [4])
  The *edit distance correction of* $w$ *with respect to* $L$, denoted $\textsc{Edc}_L(w)$, is

$$\textsc{Edc}_L(w) = \begin{cases} \text{Yes}, & \text{if } w \in L \\ \text{one string of } \{w' \in L \mid d(w, w') \text{ is minimum}\}, & \text{if } w \notin L. \end{cases}$$

The collection of all indexable classes $\mathcal{C}$ for which there is a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using MQs is denoted by *MemQ*. *EquQ*, *PCorQ*, *lBCorQ* and *EditCorQ* are defined similarly for the models of learning with EQs, PCQs, $l$-bounded correction queries ($l$BCQs) and EDCQs, respectively.

There is a strong relation between query learning models and Gold-style learning models. The following strict hierarchy holds [15]:

$$\mathit{FinTxt} \subset \mathit{FinInf} = \mathit{MemQ} \subset \mathit{ConsvTxt} \subset \mathit{LimTxt} \subset \mathit{LimInf} = \mathit{EquQ}.$$

## 3 Learning with Correction Queries

In [8], necessary and sufficient conditions for a language class to be learnable with PCQs are given, facilitating a comparison between the model of learning with PCQs and other well-known learning models. The results can be summarized as follows:

- *MemQ* is strictly included in *PCorQ*,
- *PCorQ* and *ConsvTxt* are incomparable, and
- *PCorQ* is strictly included in *LimTxt*.

We continue this study for LBCQs and EDCQs.

### 3.1 Learning with Length Bounded Correction Queries

Note that in the case of DFAs, returning the answer to a PCQ can be easily done in polynomial time. However, when the target concept ranges over arbitrary recursive languages, the answer to a PCQ might be very long or not even computable (given a recursive language $L$ and a string $w$, one cannot decide, in general, if $w$ is a prefix of a string in $L$). A possible solution to avoid very long (or infinite) searches is to restrict the search space to only short enough suffixes. So, let us consider the model in which the learner must identify the target language after asking a finite number of $l$BCQs for a fixed integer $l \geq 0$.

Since any 0BCQ can be simulated by a MQ and the other way around, it is clear that when $l = 0$, learning with $l$BCQs is equivalent to learning with MQs. It is though less straightforward that the same property holds for an arbitrary $l$. We show in the sequel that for any $l$, a language class is learnable with MQs if and only if it is learnable with $l$BCQs.

**Theorem 1.** *For any $l \geq 0$, lBCorQ = MemQ.*

*Proof.* Since for any language $L$ and any string $w$ in $\Sigma^*$, if we know the answer to $C_L^l(w)$ we also know if the string $w$ is in $L$ or not, it is clear that *lBCorQ* includes *MemQ*. Hence, we have to show only that *lBCorQ* $\subseteq$ *MemQ*.

Let us consider a language class $\mathcal{C}$ in *lBCorQ*, and let $\mathcal{A}lg$ be an $l$-bounded correction query learner for $\mathcal{C}$. We modify $\mathcal{A}lg$ such that instead of submitting an $l$BCQ for a given string $w$, to submit MQs for all the strings $wu$ with $u \in \Sigma^{\leq l}$. The learner will use this information to construct the answer for $C_L^l(w)$ (recall that $C_L^l(w) = \{u \in \Sigma^{\leq l} \mid wu \in L\}$). Clearly, this modified version of $\mathcal{A}lg$ is a query learner algorithm that learns $\mathcal{C}$ using MQs. □

This theorem is basically saying that having an oracle that can return at once the answers for more than one MQ (one $l$BCQ contains the answer for $1 + |\Sigma| + \ldots + |\Sigma|^l$ MQs) does not increase the learnability power of the model (that is, the learning with MQs model). The result was somehow expected if we recall that time complexity issues are neglected in our analysis. Moreover, this allows us to talk about the model of learning with LBCQs in general, without specifying a given length bound. Therefore, we can talk about *LBCorQ*, the collection of all language classes $\mathcal{C}$ for which there exists an $l \geq 0$ and a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using a finite number of $l$-bounded correction queries.

### 3.2 Learning with Edit Distance Correction Queries

Let us now investigate the model of learning with EDCQs. It is clear that any oracle answering EDCQs would implicitly give us the answer for the corresponding MQ, so *EditCorQ* trivially includes *MemQ*. In fact, the two learning models are equivalent:

**Theorem 2.** *EditCorQ* $=$ *MemQ*.

*Proof.* Let us first show that having an MQ oracle allows us to compute the answer to an EDCQ using a finite number of MQs. Indeed, given a non-empty recursive language $L$ and a string $w$ in $\Sigma^*$, the following algorithm computes the value of $\text{EDC}_L(w)$ by asking only MQs.

**Algorithm 1.** An algorithm that computes $\text{EDC}_L(w)$ with an MQ oracle

```
1: input: $L, w$
2: ask the oracle if $w$ is in $L$
3: if the answer is Yes then
4:    output Yes
5: else
6:    while TRUE do
7:       $i := 1$
8:       for all $u$ such that $d(w,u) = i$ do
9:          ask the oracle if $u$ is in $L$
10:         if the answer is Yes then
11:            output $u$ and halt
12:         end if
13:      end for
14:      $i := i + 1$
15:   end while
16: end if
```

Clearly, Algorithm 1 terminates by outputting a string $u \in L$ such that there is no $v \in L$ with $d(w,v) < d(w,u)$. Note that for a given $w \in \Sigma^*$ and $i \in \mathbb{N}$ there are only a finite number of strings $v \in \Sigma^*$ such that $d(w,v) = i$, and there exists an algorithm who can generate all these strings (remember that we are not concerned with the complexity of the resulting algorithm - the only requirement is to return the answer after finite steps).

Now, if we take $\mathcal{C}$ to be a language class in *EditCorQ*, then there exists an algorithm *Alg* that learns $\mathcal{C}$ using EDCQs. *Alg* can be modified to use the MQ oracle instead of the EDCQ oracle to get the necessary answers as described above. We obtained an algorithm that learns $\mathcal{C}$ using MQs only, so *EditCorQ* $\subseteq$ *MemQ* which concludes our proof. □

### 3.3 The Global Picture

A complete picture displaying the relations between all discussed versions of query learning and Gold-style learning is obtained (see Figure 1).

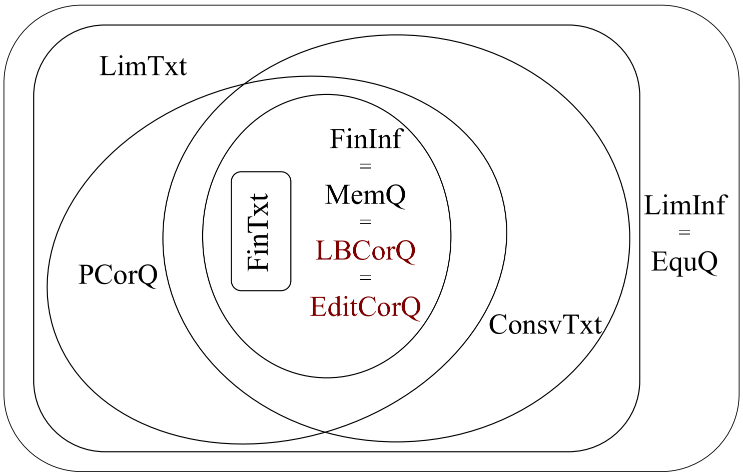


**Fig. 1.** A hierarchy of Gold-style and query learning models

As we have already mentioned, the results in this section are all about learning with queries where we do not take into account time complexity issues. So, what happens if we restrict to polynomial time learning? We will answer this question in the next section.

## 4 Polynomial Time Learning with Correction Queries

Although from a theoretical point of view it is important to know which language classes are inferable in finite time steps (see the proof of Proposition 3 for a relevant example), what matters in practice is the efficiency of the algorithms. In this section we investigate the relations between different types of CQs when complexity issues are taken into consideration. We will see that there are situations when, although two query types are equally powerful in the general case, the equality is not preserved under efficiency constraints. So far we know that learning with MQs is a strictly weaker model than learning with PCQs for both finite time algorithms [8] and polynomial ones [9]. We show by an example why one can not automatically generalize a result obtained in the general case to the restricted model of polynomial time learning.

Let us first recall that learning with EQs is strictly more powerful than learning with PCQs when ignoring time complexity: *PCorQ* is strictly included in *LimTxt* [8], and *LimTxt* is strictly included in *EquQ* [15]. Nevertheless, there exists a class of languages, namely the zero-reversible languages, that is polynomially learnable with PCQs [9] and not identifiable in polynomial time with EQs [16].

For a better comprehension of our results, let us introduce some notations. Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class. We denote by *PolMemQ* the collection of all indexable classes $\mathcal{C}$ for which there exists a polynomial $p(\cdot)$ and an algorithm $\mathcal{A}lg$

that learns any language $L$ in $\mathcal{C}$ in time $\mathcal{O}(p(size(L)))$ by asking a finite number of MQs. Similarly, *PolPCorQ*, *PollBCorQ* and *PolEditCorQ* are defined for the models of learning with PCQs, $l$BCQs and EDCQs, respectively.

## 4.1 Polynomial Time Learning with LBCQs

Let us first recall that there is basically no difference between a 0-BCQ and an MQ, so *Pol0BCorQ* = *PolMemQ*. Now, if we take $l$ to be a fixed positive integer, then the following property holds.

**Proposition 1.** *Pol(l-1)BCorQ* = *PollBCorQ for any* $l \geq 1$.

*Proof.* Since one can easily extract the answer to an $(l-1)$BCQ from the corresponding $l$BCQ, it is clear that *Pol(l-1)BCorQ* is included in *PollBCorQ*. Let us now show that the other inclusion holds as well. Let $\mathcal{C}$ be an indexed family of languages in *PollBCorQ* and $\mathcal{A}lg$ a polynomial time algorithm that learns $\mathcal{C}$ with $l$BCQs. Note that for any language $L$ over $\Sigma$, any $w \in \Sigma^*$ and any $l \geq 1$,

$$\begin{aligned} C_L^l(w) &= \{u \in \Sigma^{\leq l} \mid wu \in L\} \\ &= \{u \in \Sigma^{\leq l-1} \mid wu \in L\} \cup \{au \mid a \in \Sigma, u \in \Sigma^{l-1} \text{ and } wau \in L\} \\ &= C_L^{l-1}(w) \cup \{au \mid a \in \Sigma, u \in C_L^{l-1}(wa)\} \\ &= C_L^{l-1}(w) \cup \textstyle\bigcup_{a \in \Sigma} aC_L^{l-1}(wa). \end{aligned}$$

So, one can modify $\mathcal{A}lg$ such that instead of asking an $l$BCQ for the string $w$, to ask a finite number of $(l-1)$BCQs ($|\Sigma|+1$ queries to be precise) for the strings $wa$ with $a$ in $\{\lambda\} \cup \Sigma$. Clearly, the modified algorithm is still polynomial. Hence, *Pol(l-1)BCorQ* equals *PollBCorQ*.

We draw the conclusion that *PollBCorQ* = *PolMemQ* for any $l \geq 0$, and hence, we can talk about the model of polynomial learning with LBCQs in general, without specifying the length bound (we denote by *PolLBCorQ* the collection of all language classes $\mathcal{C}$ for which there exists an $l \geq 0$ such that $\mathcal{C}$ is in *PollBCorQ*). So, having the possibility to get answers for more than one MQ at once does not add any more learning power, even if we impose time restrictions.

## 4.2 Polynomial Time Learning with EDCQs

We continue the analysis done in Section 3.2 about the power of learning with EDCQs, this time by taking into account time complexity issues. We have seen that what happens in the general model does not necessary carry on to the polynomially bounded model. Let us recall the already known results:

– *MemQ* $\subsetneq$ *PCorQ* [8] and *PolMemQ* $\subsetneq$ *PolPCorQ* [9],
– *PCorQ* $\subsetneq$ *EquQ* [8] but *PolPCorQ* $\not\subseteq$ *PolEquQ* (see page 8 above, lines 5-8),
– *MemQ* = *LBCorQ* and *PolMemQ* = *PolLBCorQ*,
– *EditCorQ* = *MemQ*.

We show that in the case of EDCQs, the equality is not preserved.

**Proposition 2.** *PolMemQ* $\subsetneq$ *PolEditCorQ*.

*Proof.* Recall that $\mathrm{E{\small DC}}_L(w)$ is Yes if and only if $w$ belongs to $L$. Assume that $\mathcal{C}$ is a language class in *PolMemQ* and let *Alg* be a polynomial time algorithm that learns every $L$ of $\mathcal{C}$ after asking a finite number of MQs. Obviously, the number of MQs asked while *Alg* is running with input $L$ is also bounded by a polynomial, let us say $p(n)$, where $n$ is the size of the target language $L$. If we modify *Alg* so that instead of asking the oracle a MQ for the string $w$, to ask an EDCQ for the same string, we obtain another algorithm $Alg'$ which learns $\mathcal{C}$ with EDCQs (it just uses the information received from asking EDCQs to determine whether or not the given string is in the target language). The only thing left to be shown is that $Alg'$ is still polynomial. But this is clear if we notice that $Alg'$ performs at most $p(n)$ more operations than *Alg* (for each queried string $w$ it compares $\mathrm{E{\small DC}}_L(w)$ with Yes). Since $Alg'$ is a polynomial time algorithm that learns $\mathcal{C}$ using EDCQs, we obtain that $\mathcal{C}$ is in *PolEditCorQ*.

Moreover, if $\mathcal{S} = (L_w)_{w \in \Sigma^*}$ is the class of singleton languages $L_w = \{w\}$ over the alphabet $\Sigma$, then $\mathcal{S}$ can be used as a separating language class:

– $\mathcal{S} \notin$ *PolMemQ* since any algorithm that learns $\mathcal{S}$ using MQs might need to ask $|\Sigma| + |\Sigma|^2 + \ldots + |\Sigma|^{|w|}$ MQs in the worst case when running on input language $L_w$;
– $\mathcal{S} \in$ *PolEditCorQ* since there exists a very simple EDCQ algorithm for this class. Indeed, it is enough to ask one EDCQ for an arbitrarily chosen string $w$. The algorithm just outputs $w$, if the oracle's answer is Yes, and $w'$ if the answer returned by the oracle is the string $w'$. The algorithm described above is clearly polynomial in the size of the target language. □

So learning with EDCQs is strictly more powerful than leaning with MQs when we restrict to efficient algorithms.

### 4.3 The Global Picture

In the end of the previous section we exhibited a complete picture of the relations existing between several models of learning with CQs and other learning models. We have seen that when we neglect time complexity issues learning with LBCQs and EDCQs is basically the same as learning with MQs, whereas PCQs are the only ones adding some power to the model.

When we restrict to polynomial time algorithms, things are changing. And although having an LBCQs oracle does still not improve on the learnability power with respect to the MQ learning model, an EDCQ oracle or a PCQ oracle does. It is less clear what relation is between learning with EDCQs and learning with PCQs when we restrict to efficient algorithms.

Let us first notice that the class of singleton languages is in *PolPCorQ* $\cap$ *PolEditCorQ*. Moreover, we argue that there are languages polynomial time learnable with PCQs for which there is no efficient EDCQ algorithm.

**Proposition 3.** *PolPCorQ*$\setminus$*PolEditCorQ* $\neq \emptyset$.

*Proof.* From [9] we know that the class of $k$-reversible languages is in *PolPCorQ* and not in *MemQ*. But *MemQ* = *EditCorQ* by Theorem 2 (recall we mentioned that sometimes theoretical results like this one might be useful), so $k$-reversible languages are not identifiable in finite time steps with EDCQs, and hence, they can not be polynomial time learnable with EDCQs either. □

To complete the picture, we would like to be able to say if there are language classes polynomial time learnable with EDCQs for which there is no efficient PCQ algorithm. We conjecture that the two classes *PolEditCorQ* and *PolPCorQ* are incomparable (see Figure 2).

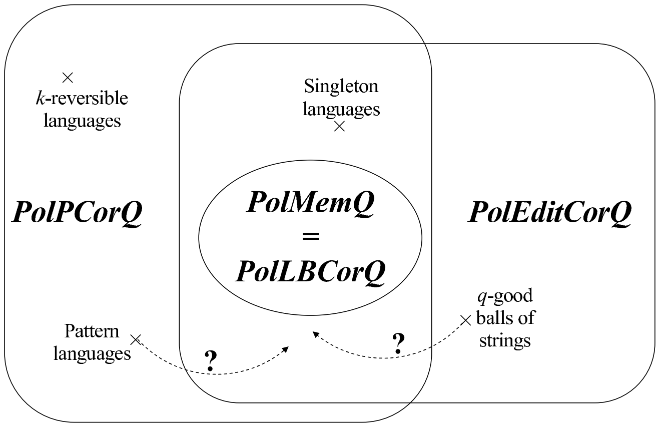


**Fig. 2.** Different types of correction queries

Our candidate for showing that $PolEditCorQ \setminus PolPCorQ \neq \emptyset$ is a subset of the class $\mathcal{B}_\Sigma = (B_r(w))_{w \in \Sigma^*, r \in \mathbb{R}}$, where $B_r(w) = \{v \in \Sigma^* \mid d(v, w) \leq r\}$ is a *ball of center $w$ and radius $r$*. The authors of [4] show that for the so-called *$q$-good balls* (the balls $B_r(w)$ for which there exists a polynomial $q(\cdot)$ such that the radius $r$ is no longer than $q(|w|)$), there exists an EDCQ algorithm which runs in polynomial time in the size of (the representation of) the language. So, what is left to be shown is that PCQs are not very useful in the process of leaning this particular class.

A slightly different type of EDCQ is used in [5] for learning the class of pattern languages: if the queried string is not in the target language, then the oracle returns the positive example with a smallest distance from the queried string and previously not used in the learning process. Moreover, preference is given to correcting strings of the same length, if any, and among those having the same length, the smallest one with respect to the lexicographical order is returned. Kinber describes in [5] an efficient algorithm that learns any pattern language with this modified type of EDCQ. We strongly believe that this requirement (i.e., that the oracle must not return as a correction any of the strings which appeared before) is actually mandatory, that is, there is no algorithm which can learn the class of pattern languages with regular EDCQs. We leave this as an open problem.

## Acknowledgements

Special thanks to Colin de la Higuera for valuable advices, and to the anonymous reviewers for their remarks and suggestions.

## References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
2. Becerra-Bonache, L., Dediu, A.H., Tîrnăucă, C.: Learning DFA from correction and equivalence queries. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 281–292. Springer, Heidelberg (2006)
3. Tîrnăucă, C.: Learning reversible languages from correction queries only, `http://grlmc-dfilrom.urv.cat/grlmc/PersonalPages/cristina/publications.htm`
4. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Learning balls of strings with correction queries. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 18–29. Springer, Heidelberg (2007)
5. Kinber, E.: On learning regular expressions and patterns via membership and correction queries (manuscript, 2008)
6. Jain, S., Kinber, E.B.: One-shot learners using negative counterexamples and nearest positive examples. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 257–271. Springer, Heidelberg (2007)
7. Tîrnăucă, C.I., Tîrnăucă, C.: Learning regular tree languages from correction and equivalence queries. Journal of Automata, Languages and Combinatorics 12(4), 501–524 (2007)
8. Tîrnăucă, C., Kobayashi, S.: A characterization of the language classes learnable with correction queries. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 398–407. Springer, Heidelberg (2007)
9. Tîrnăucă, C., Knuutila, T.: Polynomial time algorithms for learning k-reversible languages and pattern languages with correction queries. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 272–284. Springer, Heidelberg (2007)
10. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of ACM 21(1), 168–173 (1974)
11. Angluin, D.: Inference of reversible languages. Journal of the ACM 29(3), 741–765 (1982)
12. Zeugmann, T., Lange, S., Kapur, S.: Characterizations of monotonic and dual monotonic language learning. Information and Computation 120(2), 155–173 (1995)
13. Zeugmann, T.: Inductive inference and language learning. In: Cai, J.-Y., Cooper, S.B., Li, A. (eds.) TAMC 2006. LNCS, vol. 3959, pp. 464–473. Springer, Heidelberg (2006)
14. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
15. Lange, S., Zilles, S.: Formal language identification: query learning vs. Gold-style learning. Information Processing Letters 91(6), 285–292 (2004)
16. Angluin, D.: Negative results for equivalence queries. Machine Learning 5(2), 121–150 (1990)

# Unsupervised Learning of Probabilistic Context-Free Grammar Using Iterative Biclustering

Kewei Tu and Vasant Honavar

Department of Computer Science,
Iowa State University, Ames, IA 50011, USA
{tukw,honavar}@cs.iastate.edu

**Abstract.** This paper presents PCFG-BCL, an unsupervised algorithm that learns a probabilistic context-free grammar (PCFG) from positive samples. The algorithm acquires rules of an unknown PCFG through iterative biclustering of bigrams in the training corpus. Our analysis shows that this procedure uses a greedy approach to adding rules such that each set of rules that is added to the grammar results in the largest increase in the posterior of the grammar given the training corpus. Results of our experiments on several benchmark datasets show that PCFG-BCL is competitive with existing methods for unsupervised CFG learning.

## 1 Introduction

Context-free grammars (CFG) constitute an important class of grammars, with a broad range of applications including programming languages, natural language processing, and bioinformatics, among others. A probabilistic context-free grammar (PCFG) is a CFG with probabilities assigned to grammar rules, which can better accommodate the ambiguity and the need for robustness in real-world applications. Hence, the problem of learning a PCFG from data (typically, positive samples generated by the target grammar) is an important problem in grammar induction and machine learning. Several methods for learning (P)CFG from positive data have been proposed. Some rely on different heuristics to iteratively construct an approximation of the unknown CFG [1,2,3,4,5]; others search for a PCFG that has the largest posterior given the training corpus [6,7,8,9].

In this paper we propose PCFG-BCL, a new unsupervised algorithm that learns a PCFG from a positive corpus. The proposed algorithm uses (distributional) biclustering to group symbols into non-terminals. This is a more natural and robust alternative to the more widely used substitutability heuristic or distributional clustering, especially in the presence of ambiguity, e.g., when a symbol can be reduced to different nonterminals in different contexts, or when a context can contain symbols of different nonterminals, as illustrated in [1]. PCFG-BCL can be understood within a Bayesian structure search framework. Specifically, it uses a greedy approach to adding rules to a partially constructed grammar, choosing at each step a set of rules that yields the largest possible increase in the posterior of the grammar given the training corpus. The Bayesian framework also supports an ensemble approach to PCFG learning by effectively *combining* multiple candidate grammars. Results of our experiments on several benchmark datasets show that the proposed algorithm is competitive with other methods for learning CFG from positive samples.

The rest of the paper is organized as follows. Section 2 introduces the representation of PCFG used in PCFG-BCL. Section 3 describes the key ideas behind PCFG-BCL. Section 4 presents the complete algorithm and some implementation details. Section 5 presents the results of experiments. Section 6 concludes with a summary and a brief discussion of related work.

## 2 Grammar Representation

It is well-known that any CFG can be transformed into the Chomsky normal form (CNF), which only has two types of rules: $A \to BC$ or $A \to a$. Because a PCFG is simply a CFG with a probability associated with each rule, it is easy to transform a PCFG into a probabilistic version of CNF.

To simplify the explanation of our algorithm, we make use of the fact that a CNF grammar can be represented in an AND-OR form containing three types of symbols, i.e., AND, OR, and terminals. An AND symbol appears on the left-hand side of exactly one grammar rule, and on the right-hand side of that rule there are exactly two OR symbols. An OR symbol appears on the left-hand side of one or more rules, each of which has only one symbol on the right-hand side, either an AND symbol or a terminal. A multinomial distribution can be assigned to the set of rules of an OR symbol, defining the probability of each rule being chosen. An example is shown below (with rules probabilities in the parentheses).

| **CNF** | **The AND-OR Form** |
|---|---|
| $S \to a\ (0.4) \mid AB\ (0.6)$ | $\text{OR}_S \to a\ (0.4) \mid \text{AND}_{AB}\ (0.6)$ |
| $A \to a\ (1.0)$ | $\text{AND}_{AB} \to \text{OR}_A \text{OR}_B$ |
| $B \to b_1\ (0.2) \mid b_2\ (0.5) \mid b_3\ (0.3)$ | $\text{OR}_A \to a\ (1.0)$ |
| | $\text{OR}_B \to b_1\ (0.2) \mid b_2\ (0.5) \mid b_3\ (0.3)$ |

It is easy to show that a CNF grammar in the AND-OR form can be divided into a set of *AND-OR groups* plus the start rules (rules with the start symbol on the left-hand side). Each AND-OR group contains an AND symbol $N$, two OR symbols $A$ and $B$ such that $N \to AB$, and all the grammar rules that have one of these three symbols on the left-hand side. In the above example, there is one such AND-OR group, i.e., $\text{AND}_{AB}$, $\text{OR}_A$, $\text{OR}_B$ and the corresponding rules (the last three lines). Note that there is a bijection between the AND symbols and the groups; but an OR symbol may appear in multiple groups. We may simply make identical copies of such OR symbols to eliminate overlap between groups.

## 3 Main Ideas

PCFG-BCL is designed to learn a PCFG using its CNF representation in the AND-OR form. Sentences in the training corpus are assumed to be sampled from an unknown PCFG under the i.i.d. (independent and identically distributed) assumption.

Starting from only terminals, PCFG-BCL iteratively adds new symbols and rules to the grammar. At each iteration, it first learns a new AND-OR group by biclustering, as explained in Section 3.1. Once a group is learned, it tries to find rules that attach the newly learned AND symbol to existing OR symbols, as discussed in Section 3.2. This second step is needed because the first step alone is not sufficient for learning such rules. In both steps, once a new set of rules are learned, the corpus is *reduced* using the new rules, so that subsequent learning can be carried out on top of the existing learning result. These two steps are repeated until no further rule can be learned. Then start rules are added to the learned grammar in a postprocessing step (Section 3.3). Since any CNF grammar can be represented in the form of a set of AND-OR groups and a set of start rules, these three steps are capable, in principle, of constructing any CNF grammar.

We will show later that the first two steps of PCFG-BCL outlined above attempt to find rules that yield the greatest increase in the posterior probability of the grammar given the training corpus. Thus, PCFG-BCL performs a local search over the space of grammars using the posterior as the objective function.

## 3.1 Learning a New AND-OR Group by Biclustering

**Intuition.** In order to show what it means to learn a new AND-OR group, it is helpful to construct a table $T$, where each row or column represents a symbol appearing in the corpus, and the cell at row $x$ and column $y$ records the number of times the pair $xy$ appears in the corpus. Because the corpus might have been partially reduced in previous iterations, a row or column in $T$ may represent either a terminal or a nonterminal.

Since we assume the corpus is generated by a CNF grammar, there must be some symbol pairs in the corpus that are generated from AND symbols of the target grammar. Let $N$ be such an AND symbol, and let $A$, $B$ be the two OR symbols such that $N \to AB$. The set $\{x|A \to x\}$ corresponds to a set of rows in the table $T$, and the set $\{y|B \to y\}$ corresponds to a set of columns in $T$. Therefore, the AND-OR group that contains $N$, $A$ and $B$ is represented by a *bicluster*[10] (i.e., a submatrix) in $T$, and each pair $xy$ in this bicluster can be reduced to $N$. See Fig.1 (a), (b) for an example, where the AND-OR group shown in Fig.1(a) corresponds to the bicluster shown in Fig.1(b).

Further, since we assume the target grammar is a PCFG, we have two multinomial distributions defined on $A$ and $B$ respectively that independently determine the symbols generated from $A$ and $B$. Because the corpus is assumed to be generated by this PCFG, it is easy to prove that the resulting bicluster must be *multiplicatively coherent*[10], i.e., it satisfies the following condition:

$$\frac{a_{ik}}{a_{jk}} = \frac{a_{il}}{a_{jl}} \quad \text{for any two rows } i, j \text{ and two columns } k, l \tag{1}$$

where $a_{xy}$ is the cell value at row $x$ $(x = i, j)$ and column $y$ $(y = k, l)$.

$$\begin{aligned}
&\text{AND}_{NP} \to \text{OR}_{Det}\text{OR}_{N}\\
&\text{OR}_{Det} \to \text{the}(0.67) \mid \text{a}(0.33)\\
&\text{OR}_{N} \to \text{circle}(0.2)\\
&\quad \mid \text{triangle}(0.3) \mid \text{square}(0.5)
\end{aligned}$$

(a) An AND-OR group (with rule probabilities in the parentheses)

| | is | circle | triangle | square | the | ··· |
|---|---|---|---|---|---|---|
| below | | | | | 8 | |
| above | | | | | 10 | |
| the | | 24 | 36 | 60 | | |
| a | | 12 | 18 | 30 | | |
| circle | 4 | | | | | |
| triangle | 6 | | | | | |
| ⋮ | | | | | | |

(b) A part of the table $T$ and the bicluster that represents the AND-OR group. Zero cells are left blank.

| | ··· covers (.) | ··· touches (.) | ··· is above (.) | ··· is below (.) | (.) rolls. | (.) bounces. | ··· |
|---|---|---|---|---|---|---|---|
| (a, circle) | 1 | 2 | 1 | 1 | 0 | 0 | |
| (a, triangle) | 1 | 2 | 1 | 3 | 2 | 1 | |
| (a, square) | 3 | 4 | 2 | 4 | 4 | 1 | ··· |
| (the, circle) | 2 | 3 | 1 | 3 | 2 | 1 | |
| (the, triangle) | 3 | 5 | 2 | 5 | 4 | 2 | |
| (the, square) | 5 | 8 | 4 | 8 | 7 | 3 | |

(c) A part of the expression-context matrix of the bicluster

**Fig. 1.** Example: a bicluster and its expression-context matrix

Given a bicluster in $T$, we can construct an *expression-context matrix*, in which the rows represent the set of symbol pairs (expressions) in the bicluster, the columns represent all the contexts in which these symbol pairs appear, and the value in each cell denotes the number of times the corresponding expression-context combination appears in the corpus (see Fig.1(c) for an example). Because the target grammar is context-free, if a bicluster represents an AND-OR group of the target grammar, then the choice of the symbol pair is independent of its context and thus the resulting expression-context matrix should also be multiplicatively coherent, i.e., it must satisfy Eq.1.

The preceding discussion suggests an intuitive approach to learning a new AND-OR group: first find a bicluster of $T$ that is multiplicatively coherent and has a multiplicatively coherent expression-context matrix, and then construct an AND-OR group from it. The probabilities associated with the grammar rules can be estimated from the statistics of the bicluster. For example, if we find that the bicluster shown in Fig.1(b) and its expression-context matrix shown in Fig.1(c) are both multiplicatively coherent, we can learn an AND-OR group as shown in Fig.1(a).

**Probabilistic Analysis.** We now present an analysis of the intuitive idea outlined above within a probabilistic framework. Consider a trivial initial grammar where the start symbol directly generates each sentence of the corpus with equal probability. We can calculate how the likelihood of the corpus given the grammar is changed by extracting a bicluster and learning a new AND-OR group as described above.

Suppose we extract a bicluster $BC$ and add to the grammar an AND-OR group with an AND symbol $N$ and two OR symbols $A$ and $B$. Suppose there is a sentence $d$ containing a symbol pair $xy$ that is in $BC$. First, since $xy$ is reduced to $N$ after this learning process, the likelihood of $d$ is reduced by a factor of $P(N \to xy|N) = P(A \to x|A) \times P(B \to y|B)$. Second, the reduction may make some other sentences in the corpus become identical to $d$, resulting in a corresponding increase in the likelihood. Suppose the sentence $d$ is represented by row $p$ and column $q$ in the expression-context matrix of $BC$, then this second factor is exactly the ratio of the sum of column $q$ to the value of cell $pq$, because before the reduction only those sentences represented by cell $pq$ are equivalent to $d$, and after the reduction the sentences in the entire column become equivalent (the same context plus the same expression $N$).

Let $LG(BC)$ be the likelihood gain resulting from extraction of $BC$; let $G_k$ and $G_{k+1}$ be the grammars before and after extraction of $BC$, $D$ be the training corpus; in the bicluster $BC$, let $A$ denote the set of rows, $B$ the set of columns, $r_x$ the sum of entries in row $x$, $c_y$ the sum of entries in column $y$, $s$ the sum over all the entries in $BC$, and $a_{xy}$ the value of cell $xy$; in the expression-context matrix of $BC$, let EC-row denote the set of rows, EC-col the set of columns, $r'_p$ the sum of entries in row $p$, $c'_q$ the sum of entries in column $q$, $s'$ the sum of all the entries in the matrix, and $EC(p, q)$ or $a'_{pq}$ the value of cell $pq$. With a little abuse of notation we denote the context of a symbol pair $xy$ in a sentence $d$ by $d-$"xy". We can now calculate the likelihood gain as follows:

$$
\begin{aligned}
LG(BC) &= \frac{P(D|G_{k+1})}{P(D|G_k)} = \prod_{d\in D} \frac{P(d|G_{k+1})}{P(d|G_k)} \\
&= \prod_{x\in A,\ y\in B,\ xy \text{ appears in } d\in D} P(x|A)P(y|B) \frac{\sum_{p\in \text{EC-row}} EC(p, d - \text{“xy”})}{EC(\text{“xy”}, d - \text{“xy”})} \\
&= \prod_{x\in A} P(x|A)^{r_x} \prod_{y\in B} P(y|B)^{c_y} \frac{\prod_{q\in \text{EC-col}} {c'_q}^{c'_q}}{\prod_{\substack{p\in \text{EC-row} \\ q\in \text{EC-col}}} {a'_{pq}}^{a'_{pq}}}
\end{aligned}
$$

It can be shown that, the likelihood gain is maximized by setting:

$$
P(x|A) = \frac{r_x}{s} \qquad\qquad P(y|B) = \frac{c_y}{s}
$$

Substituting this into the likelihood gain formula, we get

$$
\begin{aligned}
\max_{P_r} LG(BC) &= \prod_{x\in A} \left(\frac{r_x}{s}\right)^{r_x} \prod_{y\in B} \left(\frac{c_y}{s}\right)^{c_y} \frac{\prod_{q\in \text{EC-col}} {c'_q}^{c'_q}}{\prod_{\substack{p\in \text{EC-row} \\ q\in \text{EC-col}}} {a'_{pq}}^{a'_{pq}}} \\
&= \frac{\prod_{x\in A} {r_x}^{r_x} \prod_{y\in B} {c_y}^{c_y}}{s^{2s}} \times \frac{\prod_{q\in \text{EC-col}} {c'_q}^{c'_q}}{\prod_{\substack{p\in \text{EC-row} \\ q\in \text{EC-col}}} {a'_{pq}}^{a'_{pq}}}
\end{aligned}
$$

where $P_r$ represents the set of grammar rule probabilities. Notice that $s = s'$ and $a_{xy} = r'_p$ (where row $p$ of the expression-context matrix represents the symbol pair $xy$). Thus we have

$$\max_{P_r} LG(BC) = \frac{\prod_{x \in A} {r_x}^{r_x} \prod_{y \in B} {c_y}^{c_y}}{s^s \prod_{\substack{x \in A\\ y \in B}} {a_{xy}}^{a_{xy}}} \times \frac{\prod_{p \in \text{EC-row}} {r'_p}^{r'_p} \prod_{q \in \text{EC-col}} {c'_q}^{c'_q}}{{s'}^{s'} \prod_{\substack{p \in \text{EC-row}\\ q \in \text{EC-col}}} {a'_{pq}}^{a'_{pq}}}$$

The two factors in the righthand side are of the same form, one for the bicluster and one for the expression-context matrix. This form of formula actually measures the multiplicative coherence of the underlying matrix (in a slightly different way from Eq.18 of [10]), which is maximized when the matrix is perfectly coherent. Therefore, we see that when extracting a bicluster (with the new grammar rule probabilities set to the optimal values), the likelihood gain is the product of the multiplicative coherence of the bicluster and its expression-context matrix, and that the maximal gain in likelihood is obtained when both the bicluster and its expression-context matrix are perfectly multiplicatively coherent. This validates the intuitive approach in the previous subsection. More derivation details can be found in the appendix in [11].

It must be noted however, in learning from data, simply maximizing the likelihood can result in a learned model that overfits the training data and hence generalizes poorly on data unseen during training. In our setting, maximizing the likelihood is equivalent to finding the most coherent biclusters. This can result in a proliferation of small biclusters and hence grammar rules that encode highly specific patterns appearing in the training corpus. Hence learning algorithms typically have to trade off the complexity of the model against the quality of fit on the training data. We achieve this by choosing the prior $P(G) = 2^{-DL(G)}$ over the set of candidate grammars, where $DL(G)$ is the description length of the grammar $G$. This prior penalizes more complex grammars, as complex grammars are more likely to overfit the training corpus.

Formally, the logarithm of the gain in posterior as a result of extracting an AND-OR group from a bicluster and updating the grammar from $G_k$ to $G_{k+1}$ (assuming the probabilities associated with the grammar rules are set to their optimal values) is given by:

$$\begin{aligned}
&\max_{P_r} LPG(BC) = \max_{P_r} \log \frac{P(G_{k+1}|D)}{P(G_k|D)} \\
&= \left( \sum_{x \in A} r_x \log r_x + \sum_{y \in B} c_y \log c_y - s \log s - \sum_{x \in A, y \in B} a_{xy} \log a_{xy} \right) \\
&\quad + \left( \sum_{p \in \text{EC-row}} r'_p \log r'_p + \sum_{q \in \text{EC-col}} c'_q \log c'_q - s' \log s' - \sum_{\substack{p \in \text{EC-row}\\ q \in \text{EC-col}}} a'_{pq} \log a'_{pq} \right) \\
&\quad + \alpha \left( 4 \sum_{x \in A, y \in B} a_{xy} - 2|A| - 2|B| - 8 \right) \qquad (2)
\end{aligned}$$

where $LPG(BC)$ denotes the logarithmic posterior gain resulting from extraction of the bicluster $BC$; $\alpha$ is a parameter in the prior that specifies how much the prior favors compact grammars, and hence it controls the tradeoff between the complexity of the learned grammar and the quality of fit on the training corpus. Note that the first two terms in this formula correspond to the gain in log likelihood (as shown earlier). The third term is the logarithmic prior gain, biasing the algorithm to favor large biclusters and hence compact grammars (see the appendix in [11] for details).

## 3.2 Attaching a New AND Symbol under Existing OR Symbols

**Intuition.** For a new AND symbol $N$ learned in the first step, there may exist one or more OR symbols in the current partially learned grammar, such that for each of them (denoted by $O$), there is a rule $O \to N$ in the target grammar. Such rules cannot be acquired by extracting biclusters as described above: When $O$ is introduced into the grammar, $N$ simply does not exist in the table $T$, and when $N$ is introduced, it only appears in a rule of the form $N \to AB$. Hence, we need a strategy for discovering such OR symbols and adding the corresponding rules to the grammar. Note that, if there are recursive rules in the grammar, they are learned in this step. This is because the first step establishes a partial order among the symbols, and only by this step can we connect nonterminals to form cycles and thus introduce recursions into the grammar.

Consider an OR symbol $O$ that was introduced into the grammar as part of an AND-OR group obtained by extracting a bicluster $BC$. Let $M$ be the AND symbol and $P$ the other OR symbol in the group, such that $M \to OP$. So $O$ corresponds to the set of rows and $P$ corresponds to the set of columns of $BC$.

If $O \to N$, and if we add to $BC$ a new row for $N$, where each cell records the number of appearances of $Nx$ (for all $x$ s.t. $P \to x$) in the corpus, then the expanded bicluster should be multiplicatively coherent, for the same reason that $BC$ was multiplicatively coherent. The new row $N$ in $BC$ results in a set of new rows in the expression-context matrix. This expanded expression-context matrix should be multiplicatively coherent for the same reason that the expression-context matrix of $BC$ was multiplicatively coherent. The situation is similar when we have $M \to PO$ instead of $M \to OP$ (thus a new *column* is added to $BC$ when adding the rule $O \to N$). An example is shown in Fig.2.

Thus, if we can find an OR symbol $O$ such that the expanded bicluster and the corresponding expanded expression-context matrix are both multiplicatively coherent, we should add the rule $O \to N$ to the grammar.

**Probabilistic Analysis.** The effect of attaching a new AND symbol under existing OR symbols can be understood within a probabilistic framework. Let $\widetilde{BC}$ be a *derived* bicluster, which has the same rows and columns as $BC$, but the values in its cells correspond to the expected numbers of appearances of the symbol pairs when applying the current grammar to expand the current partially reduced corpus. $\widetilde{BC}$ can be constructed by traversing all the AND symbols that $M$ can be directly or indirectly reduced to in the current grammar. $\widetilde{BC}$ is close to

AND → $OR_1OR_2$
$OR_1$ → big (0.6) | old (0.4)
$OR_2$ → dog (0.6) | cat (0.4)
**New rule:** $OR_2$ → AND

(a) An existing AND-OR group and a proposed new rule

| | dog | cat | AND |
|---|---|---|---|
| big | 27 | 18 | 15 |
| old | 18 | 12 | 10 |

(b) The bicluster and its expansion (a new column)

| | the (.) slept. | the big (.) slept. | the old (.) slept. | the old big (.) slept. | ··· heard the (.) | ··· heard the old (.) | ··· |
|---|---|---|---|---|---|---|---|
| (old, dog) | 6 | 1 | 1 | 0 | 3 | 1 | ··· |
| (big, dog) | 9 | 2 | 1 | 1 | 4 | 1 | |
| (old, cat) | 4 | 1 | 0 | 0 | 2 | 1 | |
| (big, cat) | 6 | 1 | 1 | 0 | 4 | 1 | |
| (old, AND) | 3 | 1 | 0 | 0 | 2 | 1 | ··· |
| (big, AND) | 5 | 1 | 1 | 0 | 2 | 1 | |

(c) The expression-context matrix and its expansion

**Fig. 2.** An example of adding a new rule that attaches a new AND under an existing OR. Here the new AND is attached under one of its own OR symbols, forming a self-recursion.

$BC$ if for all the AND symbols involved in the construction, their corresponding biclusters and expression-context matrices are approximately multiplicatively coherent, a condition that is ensured in our algorithm. Let $\widetilde{BC}'$ be the expanded derived bicluster that contains both $\widetilde{BC}$ and the new row or column for $N$. It can be shown that the likelihood gain of adding $O \rightarrow N$ is approximately the likelihood gain of extracting $\widetilde{BC}'$, which, as shown in Section 3.1, is equal to the product of the multiplicative coherence of $\widetilde{BC}'$ and its expression-context matrix (when the optimal new rule probabilities are assigned that maximize the likelihood gain). Thus it validates the intuitive approach in the previous subsection. See the appendix in [11] for details.

As before, we need to incorporate the effect of the prior into the above analysis. So we search for existing OR symbols that result in maximal posterior gains exceeding a user-specified threshold. The maximal posterior gain is approximated by the following formula.

$$\max_{P_r} \log \frac{P(G_{k+1}|D)}{P(G_k|D)} \approx \max_{P_r} LPG(\widetilde{BC}') - \max_{P_r} LPG(\widetilde{BC}) \qquad (3)$$

where $P_r$ is the set of new grammar rule probabilities, $G_k$ and $G_{k+1}$ is the grammar before and after adding the new rule, $D$ is the training corpus, $LPG()$ is defined in Eq.2. Please see the appendix in [11] for the details.

### 3.3 Postprocessing

The two steps described above are repeated until no further rule can be learned. Since we reduce the corpus after each step, in an ideal scenario, upon termination of this process the corpus is fully reduced, i.e., each sentence is represented by a single symbol, either an AND symbol or a terminal. However, in practice there may still exist sentences in the corpus containing more than one symbol, either because we have applied the wrong grammar rules to reduce them, or because we have failed to learn the correct rules that are needed to reduce them.

At this stage, the learned grammar is almost complete, and we only need to add the start symbol $S$ (which is an OR symbol) and start rules. We traverse the whole corpus: In the case of a fully reduced sentence that is reduced to a symbol $x$, we add $S \rightarrow x$ to the grammar if such a rule is not already in the grammar (the probability associated with the rule can be estimated by the fraction of sentences in the corpus that are reduced to $x$). In the case of a sentence that is not fully reduced, we can re-parse it using the learned grammar and attempt to fully reduce it, or we can simply discard it as if it was the result of noise in the training corpus.

## 4 Algorithm and Implementation

The complete algorithm is presented in Algorithm 1, and the three steps are shown in Algorithm 2 to 4 respectively. Algorithm 2 describes the "learning by biclustering" step (Section 3.1). Algorithm 3 describes the "attaching" step (Section 3.2), where we use a greedy solution, i.e., whenever we find a good enough OR symbol, we learn the corresponding new rule. In both Algorithm 2 and 3, a *valid* bicluster refers to a bicluster where the multiplicative coherence of the bicluster and that of its expression-context matrix both exceed a threshold $\delta$. This corresponds to the heuristic discussed in the "intuition" subsections in Section 3, and it is used here as an additional constraint in the posterior-guided search. Algorithm 4 describes the postprocessing step (Section 3.3), wherein to keep things simple, sentences not fully reduced are discarded.

**Algorithm 1.** PCFG-BCL: PCFG Learning by Iterative Biclustering

```
Input: a corpus $C$
Output: a CNF grammar in the AND-OR form
1. create an empty grammar $G$
2. create a table $T$ of the number of appearances of each symbol pair in $C$
3. repeat
4.    $G$, $C$, $T$, $N \Leftarrow$ LearningByBiclustering($G$, $C$, $T$)
5.    $G$, $C$, $T \Leftarrow$ Attaching($N$, $G$, $C$, $T$)
6. until no further rule can be learned
7. $G \Leftarrow$ Postprocessing($G$, $C$)
8. return $G$
```

**Algorithm 2.** LearningByBiclustering($G$, $C$, $T$)

```
Input: the grammar $G$, the corpus $C$, the table $T$
Output: the updated $G$, $C$, $T$; the new AND symbol $N$
 1. find the valid bicluster $Bc$ in $T$ that leads to the maximal posterior gain (Eq.2)
 2. create an AND symbol $N$ and two OR symbols $A$, $B$
 3. for all row $x$ of $Bc$ do
 4.    add $A \to x$ to $G$, with the row sum as the rule weight
 5. for all column $y$ of $Bc$ do
 6.    add $B \to y$ to $G$, with the column sum as the rule weight
 7. add $N \to AB$ to $G$
 8. in $C$, reduce all the appearances of all the symbol pairs in $Bc$ to $N$
 9. update $T$ according to the reduction
10. return $G$, $C$, $T$, $N$
```

**Algorithm 3.** Attaching($N$, $G$, $C$, $T$)

```
Input: an AND symbol $N$, the grammar $G$, the corpus $C$, the table $T$
Output: the updated $G$, $C$, $T$
 1. for each OR symbol $O$ in $G$ do
 2.    if $O$ leads to a valid expanded bicluster as well as a posterior gain (Eq.3) larger
       than a threshold then
 3.       add $O \to N$ to $G$
 4.       maximally reduce all the related sentences in $C$
 5.       update $T$ according to the reduction
 6. return $G$, $C$, $T$
```

## 4.1 Implementation Issues

In the "learning by biclustering" step we need to find the bicluster in $T$ that leads to the maximal posterior gain. However, finding the optimal bicluster is computationally intractable [10]. In our current implementation, we use stochastic hill-climbing to find only a fixed number of biclusters, from which the one with the highest posterior gain is chosen. This method is not guaranteed to find the optimal bicluster when there are more biclusters in the table than the fixed number of biclusters considered. In practice, however, we find that if there are many biclusters, often it is the case that several of them are more or less equally optimal and our implementation is very likely to find one of them.

**Algorithm 4.** Postprocessing($G$, $C$)

```
Input: the grammar $G$, the corpus $C$
Output: the updated $G$
 1. create an OR symbol $S$
 2. for each sentence $s$ in $C$ do
 3.    if $s$ is fully reduced to a single symbol $x$ then
 4.       add $S \to x$ to $G$, or if the rule already exists, increase its weight by 1
 5. return $G$
```

Constructing the expression-context matrix becomes time-consuming when the average context length is long. Moreover, when the training corpus is not large enough, long contexts often result in rather sparse expression-context matrices. Hence, in our implementation we only check context of a fixed size (by default, only the immediate left and immediate right neighbors). It can be shown that this choice leads to a matrix whose coherence is no lower than that of the true expression-context matrix, and hence may overestimate the posterior gain.

### 4.2 Grammar Selection and Averaging

Because we use stochastic hill-climbing with random start points to do biclustering, our current implementation can produce different grammars in different runs. Since we calculate the posterior gain in each step of the algorithm, for each learned grammar an overall posterior gain can be obtained, which is proportional to the actual posterior. We can use the posterior gain to evaluate different grammars and perform model selection or model averaging, which usually leads to better performance than using a single grammar.

To perform model selection, we run the algorithm multiple times and return the grammar that has the largest posterior gain. To perform model averaging, we run the algorithm multiple times and obtain a set of learned grammars. Given a sentence to be parsed, in the spirit of Bayesian model averaging, we parse the sentence using each of the grammars and use a weighted vote to accept or reject it, where the weight of each grammar is its posterior gain. To generate a new sentence, we select a grammar in the set with the probability proportional to its weight, and generate a sentence using that grammar; then we parse the sentence as described above, and output it if it's accepted, or start over if it is rejected.

## 5 Experiments

A set of PCFGs obtained from available artificial, English-like CFGs were used in our evaluation, as listed in the table below. The CFGs were converted into CNF with uniform probabilities assigned to the grammar rules. Training corpora were then generated from the resulting grammars. We compared PCFG-BCL with EMILE [1] and ADIOS [5]. Both EMILE and ADIOS produce a CFG from a training corpus, so we again assigned uniform distributions to the rules of the learned CFG in order to evaluate them.

| Grammar Name | Size (in CNF) | Recursion | Source |
|---|---|---|---|
| Num-agr | 19 Terminals, 15 Nonterminals, 30 Rules | No | Boogie[12] |
| Langley1 | 9 Terminals, 9 Nonterminals, 18 Rules | Yes | Boogie[12] |
| Langley2 | 8 Terminals, 9 Nonterminals, 14 Rules | Yes | Boogie[12] |
| Emile2k | 29 Terminals, 15 Nonterminals, 42 Rules | Yes | EMILE[1] |
| TA1 | 47 Terminals, 66 Nonterminals, 113 Rules | Yes | ADIOS[5] |

We evaluated our algorithm by comparing the learned grammar with the target grammar on the basis of *weak generative capacity*. That is, we compare the language of the learned grammar with that of the target grammar in terms of *precision* (the percentage of sentences generated by the learned grammar that are accepted by the target grammar), *recall* (the percentage of sentences generated by the target grammar that are accepted by the learned grammar), and *F-score* (the harmonic mean of precision and recall). To estimate precision and recall, 200 sentences were generated using either the learned grammar or the target grammar (as the case may be), and then parsed by the other grammar.

To ensure a fair comparison, we tuned the parameters of PCFG-BCL, EMILE and ADIOS on a separate dataset before running the evaluation experiments. Table 1 shows the experimental results. Each table cell shows the mean and standard deviation of performance estimates from 50 independent runs. In each run, each algorithm produced a single grammar as the output.

The results summarized in Table 1 show that PCFG-BCL outperformed both EMILE and ADIOS, on each of the test grammars, and by substantial margins on several of them. Moreover, in a majority of the tests, the standard deviations of the performance estimates of PCFG-BCL were lower than those of EMILE and ADIOS, suggesting that PCFG-BCL is more stable than the other two methods. It should be noted however, that neither EMILE nor ADIOS assume the training corpus to be generated from a PCFG, and thus they do not make full use of the distributional information in the training corpus. This might explain in part the superior performance of PCFG-BCL relative to EMILE and ADIOS.

We also examined the effect of grammar selection and grammar averaging (see Section 4.2), on the four datasets where PCFG-BCL did not achieve a perfect F-score on its own. In each case, we ran the algorithm for 10 times and then used the resulting grammars to perform grammar selection or grammar averaging as described in Section 4.2. The results (data not shown) show that grammar selection improved the F-score by 1.5% on average, and the largest increase of 4.4% was obtained on the TA1-200 data; grammar averaging improved the F-score by 3.2% on average, and the largest increase of 9.3% was obtained also on the TA1-200 data. In addition, both grammar selection and averaging reduced the standard deviations of the performance estimates.

**Table 1.** Experimental results. The training corpus sizes are indicated in the parentheses after the grammar names. P=Precision, R=Recall, F=F-score. The numbers in the table denote the performance estimates averaged over 50 trials, with the standard deviations in parentheses.

| Grammar | PCFG-BCL | | | EMILE | | | ADIOS | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | P | R | F | P | R | F | P | R | F |
| Num-agr (100) | 100 (0) | 100 (0) | 100 (0) | 50 (4) | 100 (0) | 67 (3) | 100 (0) | 92 (6) | 96 (3) |
| Langley1 (100) | 100 (0) | 100 (0) | 100 (0) | 100 (0) | 99 (1) | 99 (1) | 99 (3) | 94 (4) | 96 (2) |
| Langley2 (100) | 98 (2) | 100 (0) | 99 (1) | 96 (3) | 39 (7) | 55 (7) | 76 (21) | 78 (14) | 75 (14) |
| Emile2k (200) | 85 (3) | 90 (2) | 87 (2) | 75 (12) | 68 (4) | 71 (6) | 80 (0) | 65 (4) | 71 (3) |
| Emile2k (1000) | 100 (0) | 100 (0) | 100 (0) | 76 (7) | 85 (8) | 80 (6) | 75 (3) | 98 (3) | 85 (3) |
| TA1 (200) | 82 (7) | 73 (5) | 77 (5) | 77 (3) | 14 (3) | 23 (4) | 77 (24) | 55 (12) | 62 (14) |
| TA1 (2000) | 95 (6) | 100 (1) | 97 (3) | 98 (5) | 48 (4) | 64 (4) | 50 (22) | 92 (4) | 62 (17) |

## 6 Summary and Discussion

### 6.1 Related Work

Several algorithms for unsupervised learning of CFG from only positive samples are available in the literature. EMILE [1] uses a simpler form of biclustering to create new nonterminals. It performs biclustering on an initial table constructed from the unreduced corpus, finding rules with only terminals on the right-hand side; and then it turns to the substitutability heuristic to find high-level rules. In contrast, PCFG-BCL performs iterative biclustering that finds both kinds of rules. ABL [2] employs the substitutability heuristic to group possible constituents to nonterminals. Clark's algorithm [4] uses the "substitution-graph" heuristic or distributional clustering [3] to induce new nonterminals and rules. These techniques could be less robust than the biclustering method, especially in the presence of ambiguity as discussed in Section 1 and also in [1]. Both ABL and Clark's method rely on some heuristic criterion to filter non-constituents, whereas PCFG-BCL automatically identifies constituents as a byproduct of learning new rules from biclusters that maximize the posterior gain. ADIOS [5] uses a probabilistic criterion to learn "patterns" (AND symbols) and the substitutability heuristic to learn "equivalence classes" (OR symbols). In comparison, our algorithm learns the two kinds of symbols simultaneously in a more unified manner.

The inside-outside algorithm [13,14], one of the earliest algorithms for learning PCFG, assumes a fixed, usually fully connected grammar structure and tries to maximize the likelihood, making it very likely to overfit the training corpus. Subsequent work has adopted the Bayesian framework to maximize the posterior of the learned grammar given the corpus [6,7], and has incorporated grammar structure search [6,8]. Our choice of prior over the set of candidate grammars is inspired by [6]. However, compared with the approach used in [6], PCFG-BCL adds more grammar rules at each step without sacrificing completeness (the ability to find any CFG); and the posterior re-estimation in PCFG-BCL is more straightforward and efficient (by using Eq.2 and 3). An interesting recent proposal within the Bayesian framework [9] involves maximizing the posterior using a non-parametric model. Although there is no structure search, the prior used tends to concentrate the probability mass on a small number of rules, thereby biasing the learning in favor of compact grammars.

Some unsupervised methods [15,16] for learning grammatical structures other than CFG with the goal of parsing natural language sentences also employ some techniques similar to those used in CFG learning.

### 6.2 Summary and Future Work

We have presented PCFG-BCL, an unsupervised algorithm that learns a probabilistic context-free grammar (PCFG) from positive samples. The algorithm acquires rules of an unknown PCFG through iterative biclustering of bigrams in the training corpus. Results of our experiments on several benchmark datasets show that PCFG-BCL is competitive with the state of the art methods for learning CFG from positive samples. Work in progress is aimed at improving PCFG-BCL e.g., by exploring alternative strategies for optimizing the objective function, and more systematic empirical evaluation of PCFG-BCL on real-world applications (e.g., induction of grammars from natural language corpora) with respect to both weak and strong generative capacity.

## References

1. Adriaans, P., Trautwein, M., Vervoort, M.: Towards high speed grammar induction on large text corpora. In: Jeffery, K.G., Hlaváč, V., Wiedermann, J. (eds.) SOFSEM 2000. LNCS, vol. 1963. Springer, Heidelberg (2000)
2. van Zaanen, M.: Abl: Alignment-based learning. In: COLING (2000)
3. Clark, A.: Unsupervised induction of stochastic context-free grammars using distributional clustering. In: Proceedings of CoNLL (2001)
4. Clark, A.: Learning deterministic context free grammars: The omphalos competition. Machine Learning 66 (2007)
5. Solan, Z., Horn, D., Ruppin, E., Edelman, S.: Unsupervised learning of natural languages. Proc. Natl. Acad. Sci. 102(33), 11629–11634 (2005)
6. Chen, S.F.: Bayesian grammar induction for language modeling. In: Proceedings of the 33rd annual meeting on Association for Computational Linguistics (1995)
7. Kurihara, K., Sato, T.: An application of the variational bayesian approach to probabilistic contextfree grammars. In: IJCNLP 2004 Workshop beyond shallow analyses (2004)
8. Kurihara, K., Sato, T.: Variational bayesian grammar induction for natural language. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 84–96. Springer, Heidelberg (2006)
9. Liang, P., Petrov, S., Jordan, M.I., Klein, D.: The infinite pcfg using hierarchical dirichlet processes. In: Proceedings of EMNLP-CoNLL, pp. 688–697 (2007)
10. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Trans. on Comp. Biol. and Bioinformatics 1(1), 24–45 (2004)
11. Tu, K., Honavar, V.: Unsupervised learning of probabilistic context-free grammar using iterative biclustering (extended version). Technical Report 572, Computer Science, Iowa State University (2008), http://archives.cs.iastate.edu/
12. Stolcke, A.: Boogie (1993), ftp://ftp.icsi.berkeley.edu/pub/ai/stolcke/software/boogie.shar.z
13. Baker, J.K.: Trainable grammars for speech recognition. In: Speech Communication Papers for the 97th Meeting of the Acoustical Society of America (1979)
14. Lari, K., Young, S.: The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language 4, 35–36 (1990)
15. Klein, D., Manning, C.D.: Corpus-based induction of syntactic structure: Models of dependency and constituency. In: Proceedings of ACL (2004)
16. Bod, R.: An all-subtrees approach to unsupervised parsing. In: Proceedings of ACL (2006)

# Polynomial Distinguishability of Timed Automata

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen

Delft University of Technology
{S.E.Verwer,M.M.deWeerdt,C.Witteveen}@tudelft.nl

**Abstract.** We study the complexity of identifying (learning) timed automata in the limit from data. Timed automata are finite state models that model time explicitly, i.e., using numbers. Because timed automata use numbers to represent time, they can be exponentially more compact than models that model time implicitly, i.e., using states.

We show three results that are essential in order to exactly determine when timed automata are efficiently identifiable in the limit. First, we show that polynomial distinguishability is a necessary condition for efficient identifiability in the limit. Second, we prove that deterministic time automata with two or more clocks are not polynomially distinguishable. As a consequence, they are not efficiently identifiable. Last but not least, we prove that deterministic timed automata with one clock are polynomially distinguishable, which makes them very likely to be efficiently identifiable in the limit.

## 1 Introduction

*Timed automata* [1] (TAs) are finite state models that model time *explicitly*, i.e., using numbers. They can be used to model and reason about real-time systems, see e.g. [2]. In practice, it can be very difficult to construct such an automaton by hand, for instance because expert knowledge is unavailable or hard to obtain. That is why we are interested in automatically identifying such models from data. In this paper, we prove several theorems regarding the complexity of identifying TAs from data.

Often this data is obtained using sensors. This results in a time series of system states: every millisecond the state of (or event occurring in) the system is measured and recorded. From such timed data, we could have opted to identify a model that models time *implicitly*, i.e., using states. Examples of such models are the *deterministic finite state automaton* (DFA), see e.g. [3], and the *hidden Markov model* (HMM) [4]. Our main reason for modeling time explicitly is that modeling time implicitly results in an exponential blow-up of the model size: numbers use a binary representation of time while states use a unary representation of time. Because of this, we believe that if the data contains timed properties, i.e., it can be modeled efficiently using a timed automaton, then it is less efficient and much more difficult to identify an untimed model correctly from this data. In previous work [5], we have experimentally compared a state merging and transition splitting algorithm for identifying a simple timed automaton with the evidence driven state merging method (EDSM) [6] for identifying DFAs on such data. While EDSM has been the most successful method for identifying DFAs from untimed data, on timed data it performed worse than our algorithm.

In contrast to DFAs and HMMs, the identification problem for TAs has not been well-studied. We are only aware of studies on the related problem of the identification of event-recording automata (ERAs) [7]. For example, it has been shown that ERAs are identifiable in the query learning framework [8]. However, the used query learning algorithm requires an exponential amount of queries, and is hence inefficient in the amount of data it requires. Naturally, we would like our identification process to be efficient. This is difficult due to the fact the identification problem for DFAs is NP-complete [9]. This property easily generalizes to the problem of identifying a TA (by setting all time values to 0). Thus, unless $P = NP$, a TA cannot be identified efficiently. Even more troublesome is the fact that the DFA identification problem cannot even be approximated within any polynomial [10]. Hence (since this also generalizes), the TA identification problem is also inapproximable.

These two facts make the prospects of finding an efficient identification process for TAs look very bleak. However, both of these results rely on there being a fixed input for the identification problem (encoding a hard problem). While in normal decision problems this is very natural, in an identification problem the amount of input data is somewhat arbitrary: more data can be sampled if necessary. Therefore, it makes sense to study the behavior of an identification process when is it given more and more data (no longer encoding the hard problem). The framework that studies this behavior is called *identification in the limit* [11]. This framework can be summarized as follows. Let $C$ be a class of languages (for example the regular languages, modeled by DFAs). When given an increasing amount of examples from some language $L \in C$, a limit identification algorithm for $C$ should at some point converge to $L$. If there exists such an algorithm $A$, the class $C$ is said to be *identifiable in the limit*. If a polynomial amount of examples in the size of the smallest model for $L$ is sufficient for convergence of $A$, $C$ is said to be *identifiable in the limit from polynomial data*. If $A$ requires time polynomial in the size of the examples, $C$ is said to be *identifiable in polynomial time*. If both these statements hold, then $C$ is *identifiable from polynomial time and data*, i.e *efficiently identifiable in the limit*.

DFAs have been shown to be efficiently identifiable in the limit using a state merging method [12]. Also, it has been shown that *non-deterministic finite automata* (NFAs) are not efficiently identifiable in the limit [13]. This again generalizes to the problem of identifying a non-deterministic TA. Therefore, we only consider the identification problem for *deterministic timed automata* (DTAs). Our goal is to determine exactly when DTAs are efficiently identifiable in the limit and data. In this paper, we show the following results in order to achieve this goal:

– Polynomial distinguishability (Definition 6) is a necessary condition for efficient identifiability in the limit (Lemma 1).
– DTAs with two or more clocks are not polynomially distinguishable (Proposition 2). Hence, they are not efficiently identifiable (Corollary 1).
– DTAs with one clock (1-DTAs) are polynomially distinguishable (Theorem 5).

The fact that 1-DTAs are polynomially distinguishable is based on a central lemma regarding their modeling power (Lemma 2). These results tell us that 1-DTAs seem to be a good model for identifying a timed system.

The paper is organized as follows. In order to prove our results, we start with a brief introduction to DTAs (Section 2), and a formal explanation of efficient identifiability in the limit (Section 3). We then prove that DTAs are not (Section 4), and that 1-DTAs are (Section 5) polynomially distinguishable. We end our paper with a discussion regarding the obtained results (Section 6).

## 2 Timed Automata

An *timed automaton* (TA) [1] is an automaton that accepts (or generates) strings with event-time value pairs, called timed strings. A finite *timed string* $\tau$ over a finite set of symbols $\Sigma$ is a sequence $(a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ of symbol-time value pairs $(a_i, t_i) \in \Sigma \times \mathbb{N}$.[1] We use $\tau_i$ to denote the prefix of length $i$ of $\tau$, i.e., $\tau_i = (a_1, t_1) \ldots (a_i, t_i)$. Every time value $t_i$ in a timed string represents the time until the occurrence of symbol $a_i$ since the occurrence of the previous symbol $a_{i-1}$. We define the length of a timed string $\tau$, denoted $|\tau|$, as the number of symbol-time value pairs in $\tau$, i.e., $|\tau| = n$.

In TAs, timing conditions are added using a finite set $X$ of *clocks* and one *clock guard* on every transition. These clocks may have different valuations, but all move at the same speed. A *valuation* $v$ is a mapping from $X$ to $\mathbb{N}$, returning the value of a clock $x \in X$. We can add or subtract constants or other valuations to or from a valuation: if $v = v' + t$ then $\forall x \in X : v(x) = v'(x) + t$, and if $v = v' + v''$ then $\forall x \in X : v(x) = v'(x) + v''(x)$. Every transition $\delta$ in a TA is associated with a set of clocks $R$. When a transition $\delta$ occurs (or fires), the values of all the clocks in $R$ are set to 0, i.e., $\forall x \in R : v(x) := 0$. The values of all other clocks remain the same. We say that $\delta$ *resets* $x$ if $x \in R$. In this way, clocks are used to record the time since the occurrence of some specific event. Clock guards are then used to change the behavior of the TA depending on the value of clocks. A clock guard $g$ is a boolean constraint defined by the grammar: $g := x \leq c \mid x \geq c \mid g \wedge g$, where $x \in X$ is a clock, and $c \in \mathbb{N}$ is a constant.[2] A valuation $v$ is said to *satisfy* a clock guard $g$, denoted $v \in g$, if for each clock $x \in X$, whenever each occurrence of $x$ in $g$ is replaced by $v(x)$ the resulting statement is true. A timed automaton is defined as follows:

[1] Sometimes $\mathbb{R}$ is used as a time domain for TAs. However, for identification of TAs $\mathbb{N}$ is sufficient since in practice we always measure time using finite precision.

[2] Since we use the natural numbers to represent time open $(x < c)$ and closed $(x \leq c)$ timed automata are equivalent.

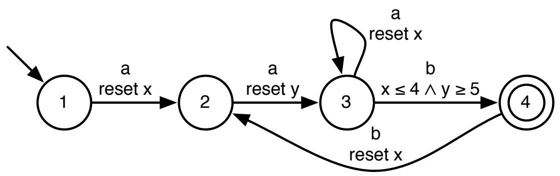


**Fig. 1.** A timed automaton. The start state (state 1) is denoted by an arrow pointing to it from nowhere. The final state (state 4) has two circles instead of one. The arrows represent transitions. The labels, clock guards, and clock resets are specified for every transition. When no guard is specified it means that the guard is always satisfied.

**Definition 1.** *A* timed automaton *(TA) is a tuple $\mathcal{A} = \langle Q, X, \Sigma, \Delta, q_0, F \rangle$, where $Q$ is a finite set of states, $X$ is a finite set of clocks, $\Sigma$ is a finite set of symbols, $\Delta$ is a finite set of transitions, $q_0$ is the start state, and $F \subseteq Q$ is a set of final states.*

*A transition $\delta \in \Delta$ is a tuple $\langle q, q', a, g, R \rangle$, where $q, q' \in Q$ are the source and target states, $a \in \Sigma$ is a symbol called the transition label, $g$ is a clock guard, and $R \subseteq X$ is the set of clock resets.*

The final states are also known as *accepting states*. The non-final states ($q \in Q \setminus F$) are known as *rejecting states*. Figure 1 shows an example of a TA. The behavior of a TA and the way it depends on time values is defined by what is called a *run* of a TA:

**Definition 2.** *A finite* run *of a TA $\mathcal{A} = \langle Q, X, \Sigma, \Delta, q_0, F \rangle$ over a (finite) timed string $\tau = (a_1, t_1) \ldots (a_n, t_n)$ is a finite sequence*

$$(q_0, v_0) \xrightarrow{t_1} (q_0, v_0 + t_1) \xrightarrow{a_1} (q_1, v_1) \xrightarrow{t_2} (q_1, v_1 + t_2) \xrightarrow{a_2} (q_2, v_2) \ldots$$
$$\ldots (q_{n-1}, v_{n-1}) \xrightarrow{t_n} (q_{n-1}, v_{n-1} + t_n) \xrightarrow{a_n} (q_n, v_n)$$

*such that for all $1 \leq i \leq n : q_i \in Q$, there exists a transition $\delta = \langle q_{i-1}, q_i, a_i, g, R \rangle \in \Delta$ such that $v_{i-1} + t_i \in g$, and for all $x \in X : v_0(x) = 0$, and $v_i(x) := 0$ if $x \in R$, $v_i(x) := v_{i-1}(x) + t_i$ otherwise.*

We call a pair $(q, v)$ of a state and a valuation a *timed state*. In a run the subsequence $(q_i, v_i + t) \xrightarrow{a_{i+1}} (q_{i+1}, v_{i+1})$ represents a state transition like in a finite automaton without time. In addition to these, a TA makes time transitions represented by $(q_i, v_i) \xrightarrow{t_{i+1}} (q_i, v_i + t_{i+1})$. A *time transition* of $t$ time units increases the value of all clocks of the TA by $t$. One can view such a transition as moving from one timed state $(q, v)$ to another timed state $(q, v+t)$ while remaining in the same untimed state $q$. We say that a timed string $\tau$ *reaches* a timed state $(q, v)$ in a TA $\mathcal{A}$ if there exist two time values $t \leq t'$ such that $(q, v') \xrightarrow{t'} (q, v' + t')$ occurs somewhere in the run of $\mathcal{A}$ over $\tau$ and $v = v' + t$. If a timed string reaches a timed state $(q, v)$ in $\mathcal{A}$ for some valuation $v$, it also reaches the untimed state $q$ in $\mathcal{A}$. A timed string *ends* in the last (timed) state it reaches, i.e., $(q_n, v_n)$

(or $q_n$). A timed string $\tau$ is *accepted* by a TA $\mathcal{A}$ if $\tau$ ends in a final state $q_f \in F$. The set of all strings $\tau$ that are accepted by $\mathcal{A}$ is called the *language* $L(\mathcal{A})$ of $\mathcal{A}$.

*Example 1.* Consider the TA $\mathcal{A}$ of Fig. 1. The run of $\mathcal{A}$ over the timed string $\tau = (a,5)(a,6)(a,2)(b,3)$ is given by: $(1,(0,0)) \xrightarrow{5} (1,(5,5)) \xrightarrow{a} (2,(0,5)) \xrightarrow{6} (2,(3,8)) \xrightarrow{a} (3,(3,0)) \xrightarrow{2} (3,(5,2)) \xrightarrow{a} (3,(0,2)) \xrightarrow{3} (3,(3,5)) \xrightarrow{b} (4,(3,5))$, where a timed state $(q,v)$ is written as $(i,(j,k))$ meaning that: $q$ is the state labeled with $i$, $v(x) = i$, and $v(y) = j$. Since state 4 is a final state, it holds that $\tau \in L(\mathcal{A})$. Note that a run cannot reach state 4 directly after reaching state 3 from state 2: the value of $x$ is greater or equal to the value of $y$ and the guard of the transition to state 4 requires it to be less then the value of $y$.

In this paper we only consider deterministic timed automata. A TA $\mathcal{A}$ is called *deterministic* (DTA) if for each possible timed string $\tau$ there exists at most one run of $\mathcal{A}$ over $\tau$. We only consider DTAs because non-deterministic TAs cannot be identified efficiently in the limit due to the fact that untimed non-deterministic automata are not efficiently identifiable in the limit [13].

## 3 Efficient Identification in the Limit

An identification process tries to find (learn) a model that explains a set of observations (data). The ultimate goal of such a process is to find a model equivalent to the actual concept that was responsible for producing the observations, called the *target concept*. In our case, we try to find a DTA model $\mathcal{A}$ that is equivalent to a target language $L_t$, i.e., $L(\mathcal{A}) = L_t$. If this is the case, we say that $L_t$ is identified correctly. We try to find this model using *labeled data*: an *input sample* $S$ is a pair of finite sets of positive examples $S_+ \subseteq L_t$ and negative examples $S_- \subseteq L_t^c = \{\tau \mid \tau \notin L_t\}$. We modify the non-strict set inclusion operators for input samples such that they operate on the positive and negative examples separately, for example if $S = (S_+, S_-)$ and $S' = (S'_+, S'_-)$ then $S \subseteq S'$ means $S_+ \subseteq S'_+$ and $S_- \subseteq S'_-$.

An identification process is called *efficient in the limit* (from polynomial time and data) if the time and data it needs to converge to the target concept are both polynomial in the size of the target concept. Efficient identifiability in the limit can be proved by showing the existence of polynomial *characteristic sets* [13].

**Definition 3.** *A* characteristic set $S_{cs}$ *of a target language* $L_t$ *for an identification algorithm* $A$ *is an input sample* $\{S_+ \in L_t, S_- \in L_t^c\}$ *such that:*

- *given* $S_{cs}$ *as input, algorithm* $A$ *identifies* $L_t$ *correctly, i.e.,* $A$ *returns an automaton* $\mathcal{A}$ *such that* $L(\mathcal{A}) = L_t$,
- *and given any input sample* $S' \supseteq S_{cs}$ *as input, algorithm* $A$ *still identifies* $L_t$ *correctly.*[3]

[3] This requirement is necessary to avoid collusion: otherwise it is possible to encode $L_t$ in $S'$ making identification a trivial task.

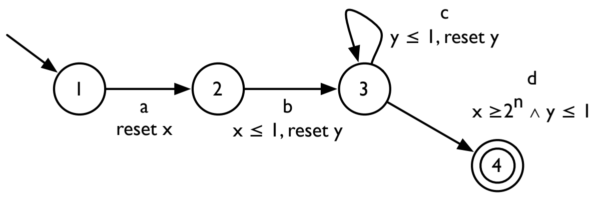


**Fig. 2.** In order to reach state 4, we require a string of exponential length ($2^n$). However, due to the binary encoding of clock guards, the DTA is of size polynomial in $n$.

**Definition 4.** *A class of automata $C$ is* efficiently identifiable in the limit *if there exist two polynomials $p$ and $q$ and an algorithm $A$ such that:*

- *given an input sample of size $n$, $A$ runs in time bounded by $p(n)$,*
- *and for every target language $L_t = L(\mathcal{A})$, $\mathcal{A} \in C$, there exists a characteristic set $S_{cs}$ of $L_t$ for $A$ of size bounded by $q(|\mathcal{A}|)$.*

## 4 Timed Automata Are Not Efficiently Identifiable

DTAs are not efficiently identifiable in the limit. The reason is that in order to reach some parts of a DTA, one may need a timed string of exponential length. We give an example of this in Fig. 2. Formally, this example can be used to show that in general DTAs are not *polynomially reachable*:

**Definition 5.** *We call a class of automata $C$* polynomially reachable *if there exists a polynomial function $p$, such that for any reachable state $q$ from any $\mathcal{A} \in C$, there exists a string $\tau$, with $|\tau| \leq p(|\mathcal{A}|)$, such that $\tau$ reaches $q$ in $\mathcal{A}$.*

**Proposition 1.** *The class of DTAs is not polynomially reachable.*

*Proof.* Let $C^* = \{\mathcal{A}_n \mid n \geq 1\}$ denote the (infinite) class of DTAs defined by Fig. 2. In any DTA $\mathcal{A}_n \in C^*$, state $q = 4$ can be reached only if both $x \geq 2^n$ and $y \leq 1$ are satisfied. Moreover, $x \leq 1$ is satisfied when $y$ is reset for the first time, and later $y$ can be reset only if $y \leq 1$ is satisfied. Therefore, in order to satisfy both $y \leq 1$ and $x \geq 2^n$, $y$ has to be reset $2^n$ times. Hence, the shortest string $\tau$ that reaches state $q = 4$ is of length $2^n$. However, since the clock guards are encoded in binary, the size of $\mathcal{A}_n$ is only polynomial in $n$. Thus, there exists no polynomial function $p$ such that $\tau \leq p(|\mathcal{A}_n|)$. Since every $\mathcal{A}_n \in C^*$ is a DTA, DTAs are not polynomially reachable.

The non-polynomial reachability of DTAs implies non-polynomial distinguishability of DTAs:

**Definition 6.** *We call a class of automata $C$* polynomially distinguishable *if there exists a polynomial function $p$, such that for any two automata $\mathcal{A}, \mathcal{A}' \in C$ such that $L(\mathcal{A}) \neq L(\mathcal{A}')$, there exists a string $\tau \in L(\mathcal{A}) \triangle L(\mathcal{A}')$, such that $|\tau| \leq p(|\mathcal{A}| + |\mathcal{A}'|)$.*

**Proposition 2.** *The class of DTAs is not polynomially distinguishable.*

*Proof.* DTAs are not polynomially reachable, hence there exists no polynomial function $p$ such that for every state $q$ of any DTA $\mathcal{A}$, the length of the shortest timed string $\tau$ that reaches $q$ in $\mathcal{A}$ is bounded by $p(|\mathcal{A}|)$. So, there is a DTA $\mathcal{A}$ with a state $q$ for which the length of $\tau$ cannot be polynomially bounded by $p(|\mathcal{A}|)$. Given this $\mathcal{A} = \langle Q, X, \Sigma, \Delta, q_0, F \rangle$, construct two DTAs $\mathcal{A}_1 = \langle Q, X, \Sigma, \Delta, q_0, \{q\} \rangle$ and $\mathcal{A}_2 = \langle Q, X, \Sigma, \Delta, q_0, \emptyset \rangle$. By definition of $\mathcal{A}_1$ and $\mathcal{A}_2$, $\tau$ is the shortest string such that $\tau \in L(\mathcal{A}_1) \triangle L(\mathcal{A}_2)$. Since $|\mathcal{A}_1| + |\mathcal{A}_2| \leq 2 \times |\mathcal{A}|$, there exists no polynomial function $p$ such that the length of $\tau$ is bounded by $p(|\mathcal{A}_1| + |\mathcal{A}_2|)$. Hence the class of DTAs is not polynomially distinguishable.

It is fairly straightforward to show that polynomial distinguishability is a necessary requirement for efficient identifiability:

**Lemma 1.** *If a class of automata $C$ is efficiently identifiable, then $C$ is polynomially distinguishable.*

*Proof.* Suppose a class of automata $C$ is efficiently identifiable, but not polynomially distinguishable. Thus, there exists no polynomial function $p$ such that for any two automata $\mathcal{A}, \mathcal{A}' \in C$ (with $L(\mathcal{A}) \neq L(\mathcal{A}')$) the length of the shortest timed string $\tau \in L(\mathcal{A}) \triangle L(\mathcal{A}')$ is bounded by $p(|\mathcal{A}| + |\mathcal{A}'|)$. Let $\mathcal{A}$ and $\mathcal{A}'$ be two automata for which such a function $p$ does not exist and let $S_{cs}$ and $S'_{cs}$ be their polynomial characteristic sets. Let $S = S_{cs} \cup S'_{cs}$ be the input sample for the identification algorithm $A$ for $C$ from Definition 4. Since $C$ is not polynomially distinguishable, neither $S_{cs}$ or $S'_{cs}$ contains an example $\tau$ such that $\tau \in L(\mathcal{A})$ and $\tau \notin L(\mathcal{A}')$, or vice versa (because no distinguishing string is of polynomial length). Hence, $S = (S_+, S_-)$ is such that $S_+ \subseteq L(\mathcal{A})$, $S_+ \subseteq L(\mathcal{A}')$, $S_- \subseteq L(\mathcal{A})^{\mathrm{C}}$, and $S_- \subseteq L(\mathcal{A})^{\mathrm{C}}$. The second requirement of Definition 3 now requires that $A$ returns both $\mathcal{A}$ and $\mathcal{A}'$, a contradiction.

This leads to the main result of this section:

**Theorem 1.** *DTAs cannot be identified efficiently.*

*Proof.* By Proposition 2 and Lemma 1.

Or more specifically:

**Corollary 1.** *DTAs with two or more clocks cannot be identified efficiently.*

*Proof.* The corollary follows from the fact that the argument of Proposition 1 only requires a DTA with two clocks.

This result seems to shatter all hope of ever finding an efficient algorithm for identifying DTAs. Instead of identifying general DTAs, we therefore would like to focus on subclasses of DTAs that are efficiently identifiable.

## 5 Polynomially Distinguishable Timed Automata

In the previous section we showed DTAs not to be efficiently identifiable in general. The proof for this result was based on the fact that DTAs are not polynomially distinguishable. Since polynomial distinguishability is a necessary requirement for efficient identifiability, we are interested in classes of DTAs that are polynomially distinguishable. In this section, we show that DTAs with a single clock are polynomially distinguishable.

A one-clock DTA (1-DTA) is a DTA that contains exactly one clock, i.e., $|X| = 1$. Our proof that 1-DTAs are polynomially distinguishable is based on the following observation:

– If a timed string $\tau$ reaches some timed state $(q, v)$ in a 1-DTA $\mathcal{A}$, then all timed states $(q, v')$ with $v'(x) \geq v(x)$ can be reached in $\mathcal{A}$.

This holds because when a timed string reaches $(q, v)$ it could have made a bigger time transition to reach all bigger valuations. This property is specific to 1-DTAs: a DTA with multiple clocks can wait in $q$, but only those bigger valuations can be reached where the difference between the clocks remains the same. It is this property of 1-DTAs that allows us to polynomially bound the length of a timed string that distinguishes between two 1-DTAs. We first use this property to show that 1-DTAs are polynomially reachable. We then use a similar argument to show the polynomial distinguishability of 1-DTAs.

**Proposition 3.** *1-DTAs are polynomially reachable.*

*Proof.* Given a 1-DTA $\mathcal{A} = \langle Q, \{x\}, \Sigma, \Delta, q_0, F\rangle$, let $\tau = (a_1, t_1) \dots (a_n, t_n)$ be a shortest timed string such that $\tau$ reaches some state $q_n \in Q$. Suppose that some prefix $\tau_i = (a_1, t_1) \dots (a_i, t_i)$ of $\tau$ ends in some timed state $(q, v)$. Then for any $j > i$, $\tau_j$ cannot end in $(q, v')$ if $v(x) \leq v'(x)$. If this were the case, $\tau_i$ instead of $\tau_j$ could be used to reach $(q, v')$, and hence a shorter timed string could be used to reach $q_n$, resulting in a contradiction. Thus, for some index $j > i$, if $\tau_j$ also ends in $q$, then $x$ has to be reset between index $i$ and $j$ in $\tau$. In other words, there exists some index $i < k \leq j$ and a state $q' \neq q$ such that $\tau_k$ ends in $(q', v_0)$, where $v_0(x) = 0$. It follows that, if $x$ is reset at index $i$ ($\tau_i$ ends in $(q, v_0)$), there cannot exist any index $j > i$ such that $\tau_j$ ends in $q$. Hence:

– For every state $q \in Q$, the number of prefixes of $\tau$ that end in $q$ is bounded by the number of times $x$ is reset by $\tau$.
– For every state $q' \in Q$, there exists at most one index $i$ such that $\tau_i$ ends in $(q', v_0)$. In other words, $x$ is reset by $\tau$ at most $|Q|$ times.

Consequently, each state is visited at most $|Q|$ times by the run of $\mathcal{A}$ on $\tau$. Thus, the length of a shortest timed string $\tau$ that reaches $q_n$ is bounded by $|Q| * |Q|$, which is polynomial in the size of $\mathcal{A}$.

Given that 1-DTAs are polynomially reachable, one would guess that it should be easy to prove the polynomial distinguishability of 1-DTAs. However, this is not the case. The main problem is that when considering the difference between two 1-DTAs, we effectively have access to two clocks instead of one. Note that, although we have access to two clocks, there are no clock guards that bound both clock values. Because of this, we cannot construct DTAs such as the one in Fig. 2. Our proof for the polynomial distinguishability of 1-DTAs follows the same line of reasoning as our proof of Proposition 3, although it is much more complicated to bound the amount of times $x$ is reset. We have split the proof of this bound into several proofs of smaller propositions and lemmas. The main theorem follows from combining these propositions and lemmas.

For the remainder of this section, let $\mathcal{A}_1 = \langle Q_1, \{x_1\}, \Sigma_1, \Delta_1, q_{1,0}, F_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \{x_2\}, \Sigma_2, \Delta_2, q_{2,0}, F_2 \rangle$ be two 1-DTAs. Let $\tau = (a_1, t_1) \ldots (a_n, t_n)$ be a shortest string that distinguishes between these 1-DTAs, i.e., $\tau \in L(\mathcal{A}_1)$, $\tau \notin L(\mathcal{A}_2)$, or vice versa, and the size of $\tau$ is minimal amongst all such timed strings. The *combined run* of $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\tau$ is the sequence:

$$\langle q_{1,0}, v_{1,0}, q_{2,0}, v_{2,0} \rangle \xrightarrow{t_1} \langle q_{1,0}, v_{1,0} + t_1, q_{2,0}, v_{2,0} + t_1 \rangle \ldots$$
$$\ldots \langle q_{1,n-1}, v_{1,n-1} + t_n, q_{2,n-1}, v_{2,n-1} + t_n \rangle \xrightarrow{a_n} \langle q_{1,n}, v_{1,n}, q_{2,n}, v_{2,n} \rangle$$

where $(q_{1,0}, v_{1,0}) \xrightarrow{t_1} (q_{1,0}, v_{1,0} + t_1) \ldots (q_{1,n-1}, v_{1,n-1} + t_n) \xrightarrow{a_n} (q_{1,n}, v_{1,n})$ is the run of $\mathcal{A}_1$ over $\tau$ and $(q_{2,0}, v_{2,0}) \xrightarrow{t_1} (q_{2,0}, v_{2,0} + t_1) \ldots (q_{2,n-1}, v_{2,n-1} + t_n) \xrightarrow{a_n}$ $(q_{2,n}, v_{2,n})$ is the run of $\mathcal{A}_2$ over $\tau$. All the definitions of properties of runs are easily adapted to properties of combined runs. We now use the notion of a combined run to show the following:

**Proposition 4.** *The length of $\tau$ is bounded by a polynomial in the size of $\mathcal{A}_1$, the size of $\mathcal{A}_2$, and the number of times $x_1$ or $x_2$ is reset by $\tau$.*

*Proof.* Suppose that for some index $1 \leq i \leq n$, $\tau_i$ ends in $\langle q_1, v_1, q_2, v_2 \rangle$. Using the same argument used in the proof of Proposition 3, one can show that for every $j > i$ and for some $v_1'$ and $v_2'$, if $\tau_j$ ends in $\langle q_1, v_1', q_2, v_2' \rangle$, then there exists an index $i < k \leq j$ such that $\tau_k$ ends in $(q_1', v_{1,0})$ in $\mathcal{A}_1$ for some $q_1' \in Q_1$, or in $(q_2', v_{2,0})$ in $\mathcal{A}_2$ for some $q_2' \in Q_2$. Thus, for every combined state $(q_1, q_2) \in Q_1 \times Q_2$, the number of prefixes of $\tau$ that end in $(q_1, q_2)$ is bounded by the number of times $r$ that $\tau$ resets either $x_1$ or $x_2$. Hence the length of $\tau$ is bounded by $|Q_1| * |Q_2| * r$, which is polynomial in $r$ and in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$.

We want to bound the number of clock resets in the combined run of a shortest distinguishing string $\tau$. In order to do so, we first prove a restriction on the possible clock valuations in a combined state $(q_1, q_2)$ that is reached directly after one clock $x_1$ has been reset. In Proposition 3, there was exactly one possible valuation, namely $v(x) = 0$. But since we now have an additional clock $x_2$, this restriction no longer holds. We can show, however, that after the second time $(q_1, q_2)$ is reached by $\tau$ directly after resetting $x_1$, the valuation of $x_2$ has to be decreasing with respect to the previous times $\tau$ reached $(q_1, q_2)$:

**Lemma 2.** *If there exists (at least) three indexes $1 \leq i < j < k \leq n$ such that $\tau_i$ ends in $\langle q_1, v_{1,0}, q_2, v_{2,i} \rangle$, $\tau_j$ ends in $\langle q_1, v_{1,0}, q_2, v_{2,j} \rangle$, and $\tau_k$ ends in $\langle q_1, v_{1,0}, q_2, v_{2,k} \rangle$, then the valuation of $x_2$ has to be decreasing, i.e., it has to be the case that $v_{2,i}(x_2) > v_{2,k}(x_2)$ and $v_{2,j}(x_2) > v_{2,k}(x_2)$.*

*Proof.* Without loss of generality we assume that $\tau \in L(\mathcal{A}_1)$, and consequently $\tau \notin L(\mathcal{A}_2)$. Let $l = k + 1$, and let $\tau_{-l} = (a_{l+1}, t_{l+1}) \dots (a_n, t_n)$ denote the suffix of $\tau$ starting at index $l + 1$, i.e., $\tau = \tau_k(a_l, t_l)\tau_{-l}$. Assume for the sake of contradiction that the valuations $v_{2,i}$, $v_{2,j}$, and $v_{2,k}$ are such that $v_{2,i}(x_2) < v_{2,j}(x_2) < v_{2,k}(x_2)$ (the argument below can be repeated for the case when $v_{2,j}(x_2) < v_{2,i}(x_2) < v_{2,k}(x_2)$). Let $d_1$ and $d_2$ denote the differences in clock values of $x_2$ between the first and second, and second and third time $(q_1, q_2)$ is reached by $\tau$, i.e., $d_1 = v_{2,j}(x_2) - v_{2,i}(x_2)$ and $d_2 = v_{2,k}(x_2) - v_{2,j}(x_2)$.

We are now going to make some observations about the acceptance property of the runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\tau$. But, instead of following the path specified by $\tau$, we are going to make a time transition in $(q_1, q_2)$ and then only run the final part of $\tau$. Under our assumption, this is possible because we assume that $(q_1, q_2)$ is reached (at least) three times, and each time the valuation of $x_2$ is increasing. Hence, in $\mathcal{A}_2$ we can reach the timed state that is reached the last time $(q_1, q_2)$ is visited (at index $k$) by making a time transition. Because we reach the same timed state and the subsequent run is identical, the acceptance property has to remain the same. We know that $\tau = \tau_k(a_l, t_l)\tau_{-l} \notin L(\mathcal{A}_2)$. Hence, it has to hold that $\tau_j(a_l, t_l + d_2)\tau_{-l} \notin L(\mathcal{A}_2)$ and that $\tau_i(a_l, t_l + d_1 + d_2)\tau_{-l} \notin L(\mathcal{A}_2)$. Similarly, since $\tau \in L(\mathcal{A}_1)$, and since $\tau_i$, $\tau_j$, and $\tau_k$ all end in the same timed state $(q_1, v_{1,0})$ in $\mathcal{A}_1$, it holds that $\tau_i(a_l, t_l)\tau_{-l} \in L(\mathcal{A}_1)$ and that $\tau_j(a_l, t_l)\tau_{-l} \in L(\mathcal{A}_1)$. Lets put this type of information in a table (+ denotes true, and − denotes false):

| value of $t$ | $0$ | $d_1$ | $d_2$ | $(d_1 + d_2)$ |
|---|---|---|---|---|
| $\tau_i(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_1)$ | + | | | |
| $\tau_i(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_2)$ | | | | − |
| $\tau_j(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_1)$ | + | | | |
| $\tau_j(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_2)$ | | | − | |

Since $\tau$ is a shortest distinguishing string, it cannot be that both $\tau_i(a_l, t_l+t)\tau_{-l} \in L(\mathcal{A}_1)$ and $\tau_i(a_l, t_l + t)\tau_{-l} \notin L(\mathcal{A}_2)$ (or vice versa) hold for for some $t \in \mathbb{N}$. Otherwise, $\tau_i(a_l, t_l + t)\tau_{-l}$ would be a shorter distinguishing string for $\mathcal{A}_1$ and $\mathcal{A}_2$. This also holds if we replace $i$ by $j$. Furthermore, since $\tau_i$ ends in the same timed state as $\tau_j$ in $\mathcal{A}_2$, it holds that for all $t \in \mathbb{N}$: $\tau_i(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_2)$ if and only if $\tau_j(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_2)$. The table thus becomes:

| value of $t$ | $0$ | $d_1$ | $d_2$ | $(d_1 + d_2)$ |
|---|---|---|---|---|
| $\tau_i(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_1)$ | + | | − | − |
| $\tau_i(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_2)$ | + | | − | − |
| $\tau_j(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_1)$ | + | | − | − |
| $\tau_j(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_2)$ | + | | − | − |

Now, since $\tau_i(a_l, t_l + d_1)$ ends in the same timed state as $\tau_j(a_l, t_l)$ in $\mathcal{A}_1$, it holds that $\tau_i(a_l, t_l + d_1)\tau_{-l} \in L(\mathcal{A}_1)$ if and only if $\tau_j(a_l, t_l)\tau_{-l} \in L(\mathcal{A}_1)$ (they reach the same timed state and then their subsequent runs are identical). More generally, for any time value $t \in \mathbb{N}$, $\tau_i(a_l, t_l + d_1 + t)\tau_{-l} \in L(\mathcal{A}_1)$ if and only if $\tau_j(a_l, t_l + t)\tau_{-l} \in L(\mathcal{A}_1)$. Hence we can extend the table in the following way:

| value of $t$ | $0$ | $d_1$ | $d_2$ | $(d_1+d_2)$ | $2d_1$ | $(2d_1+d_2)$ | $3d_1$ | $(3d_1+d_2)$ |
|---|---|---|---|---|---|---|---|---|
| $\tau_i(a_l, t_l+t)\tau_{-l} \in L(\mathcal{A}_1)$ | $+$ | $+$ | $-$ | $-$ | $+$ | $-$ | $+$ | $-$ |
| $\tau_i(a_l, t_l+t)\tau_{-l} \in L(\mathcal{A}_2)$ | $+$ | $+$ | $-$ | $-$ | $+$ | $-$ | $+$ | $-$ |
| $\tau_j(a_l, t_l+t)\tau_{-l} \in L(\mathcal{A}_1)$ | $+$ | $+$ | $-$ | $-$ | $+$ | $-$ | $+$ | $-$ |
| $\tau_j(a_l, t_l+t)\tau_{-l} \in L(\mathcal{A}_2)$ | $+$ | $+$ | $-$ | $-$ | $+$ | $-$ | $+$ | $-$ |

This extention can be continued infinitely. Thus, for any value $n \in \mathbb{N}$ it holds that $\tau_i(a_l, t_l + n * d_1)\tau_{-l} \in L(\mathcal{A}_1)$ and $\tau_i(a_l, t_l + n * d_1 + d_2)\tau_{-l} \notin L(\mathcal{A}_1)$. This can only be the case if a different transition is fired for each of these $|\mathbb{N}|$ different values for $x$. Consequently, $\mathcal{A}$ should contain an infinite amount of transitions, and hence $\mathcal{A}$ is not a 1-DTA, a contradiction.

We have just shown that if $\tau$ reaches some combined state $(q_1, q_2)$ twice directly after a reset of $x_1$ (i.e., $\langle q_1, v_{1,0}, q_2, v\rangle$ is reached at least twice for some $v$), then the valuation of $x_2$ has to be *decreasing* with respect to the previous time it reached $(q_1, q_2)$. Without loss of generality we assume that $x_1$ has already been reset at least twice at previous indexes just before reaching $(q_1, q_2)$. This can be used to show that if $\tau$ reaches $(q1, q_2)$ again then:

- $x_2$ is reset before again reaching $(q_1, q_2)$ and resetting $x_1$, and
- on the path from $(q_1, q_2)$ to $(q_1, q_2)$, there has to exist at least one transition that cannot be satisfied by a valuation smaller than the one reached by $\tau$.

The first statement follows from the observation that the valuation of $x_2$ has to be decreasing. The second statement holds because if there is no such transition, then a timed string $\tau'$ exists that reaches a smaller valuation of $x_2$ than $\tau$ when it reaches $(q_1, q_2)$ again. By the argument of Lemma 2, it cannot be the case that a non-shortest distinguishing reaches a smaller valuation than $\tau$. Hence this either leads to a contradiction or $\tau'$ is a shortest distinguishing string. In this case the statement holds for the shortest distinguishing string $\tau'$. Formally:

**Corollary 2.** *If for some index $i$, $\tau_i$ ends in $(q_1, q_2)$ directly after a reset of $x_1$, and if there exists an index $j > i$ such that $\tau_j$ ends in $(q_1, q_2)$ directly after a reset of $x_1$, then there exists an index $i < k \leq j$ such that $\tau_k$ ends in $(q_{2,k}, v_{2,0})$.*

*Proof.* By Lemma 2, it holds that $v_{2,i}(x_2) > v_{2,j}(x_2)$. The value of $x_2$ can only decrease if it is reset. Hence, there exists an index $i < k \leq j$ at which $x_2$ is reset.

**Corollary 3.** *If for some index $i$, $\tau_i$ ends in $(q_1, q_2)$ directly after a reset of $x_1$, and if there exists another index $j > i$ such that $\tau_j$ ends in $(q_1, q_2)$ directly after a reset of $x_1$, then there exists an index $i < l \leq j$ such that $\tau_l$ ends in $\langle q_{1,l}, v_{1,l}, q_{2,l}, v_{2,l}\rangle$, where either $v_{1,l}$ or $v_{2,l}$ is the minimum clock valuation satisfying the last transition fired by $\tau_l$ in $\mathcal{A}_1$ or $\mathcal{A}_2$, respectively.*

*Proof.* Suppose there exists no such index $l$. In this case, we can subtract 1 from a time value occurring in $\tau_j$ to create a timed string $\tau'_j$ that follows the same path as $\tau_j$, but ends in $\langle q_1, v_{1,0}, q_2, v_{2,j} - 1\rangle$ instead of $\langle q_1, v_{1,0}, q_2, v_{2,j}\rangle$. Without loss of generality $\tau'_j$ is not a shortest distinguishing string (otherwise the corollary also holds). We know that $\tau'_j$ reaches a smaller valuation than $\tau_l$, i.e., it holds that $v_{2,j}(x_2) < v_{2,j}(x_2)$. Hence $\tau'_j$ can be used to reach a contradiction using the argument of Lemma 2.

We now use these these two properties of $\tau$ to polynomially bound the number of different ways in which $x_2$ can be reset by $\tau$ before reaching $(q_1, q_2)$. By Corollary 2, this also polynomially bounds the amount of resets of $x_1$. In combination with Proposition 4 this proves that 1-DTAs are polynomially distinguishable.

**Lemma 3.** *The number of times $x_2$ is reset by $\tau$ before reaching a combined state $(q_1, q_2)$ directly after a reset of $x_1$ is bounded by a polynomial in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$*

*Proof.* Suppose $x_1$ is reset at index $1 \leq i \leq n$ just before reaching $(q_1, q_2)$, i.e., $\tau_i$ ends in $\langle q_1, v_{1,0}, q_2, v_{2,i}\rangle$. Thus, by Lemma 2, $v_{2,i}$ is decreasing with respect to previous indexes. By Corollary 2, we know that $x_2$ has reset before index $i$. Let $k < i$ be the largest index before $i$ where $x_2$ is reset. By Lemma 2, we know that $v_{1,k}$ is decreasing with respect to previous indexes. We also know, by Corollary 3, that there exists an index $l$ such that the last transition that is fired $\tau_l$ in either $\mathcal{A}_1$ or $\mathcal{A}_2$ has a clock guard $g_1$ or $g_2$ such that the minimal valuation that satisfies $g_1$ or $g_2$ is $v_{1,l}$ or $v_{2,l}$, respectively. Let us consider these two cases.

Suppose $v_{1,l}$ is the minimal valuation that satisfies $g_1$. Since $v_{1,k}$ is decreasing, and $x_1$ is not reset between index $k$ and $l$, it has to be the case that $v_{1,l}$ is decreasing. Hence, if at later indexes $i < m < o$ it again occurs that: $x_1$ is reset at index $o$ just before reaching $(q_1, q_2)$, $m$ is the largest index before $o$ where $x_2$ is reset, and $\tau_m$ ends in the same combined state as $\tau_k$, then there can be no index $p$ such that $v_{1,p}$ satisfies $g_1$. Thus, if the same combined state $(q'_1, q'_2)$ is used to reset $x_2$ before reaching $(q_1, q_2)$ and resetting $x_1$, there exists at least one transition in $\mathcal{A}_1$ that can no longer be fired on the path from $(q'_1, q'_2)$ to $(q_1, q_2)$. Hence, there are at most $|Q_1| * |Q_2| * |\Delta_1|$ ways in which this can occur in $\tau$.

Suppose $v_{2,l}$ is the minimal valuation that satisfies $g_2$. Since $v_{2,i}$ is decreasing, and $x_2$ is not reset between index $l$ and $i$, it has to be the case that $v_{2,l}$ is decreasing. Hence if at some later index $i < o$ it occurs again that $x_1$ is reset at index $o$ just before reaching $(q_1, q_2)$, then there can be no index $p$ such that $v_{2,p}$ satisfies $g_2$. Hence, there are at most $|\Delta_2|$ ways in which this can occur.

In conclusion, the number of times that $x_2$ can be reset by $\tau$ before reaching $(q_1, q_2)$ directly after a reset of $x_1$ is bounded by $(|Q_1| * |Q_2| * |\Delta_1| + |\Delta_2|)$, which is polynomial in the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$.

We are now ready to show the main result of this section:

**Theorem 2.** *1-DTAs are polynomially distinguishable.*

*Proof.* By Lemma 2, after the second time a combined state $(q_1, q_2)$ is reached by $\tau$ after resetting $x_1$, it can only be reached again if $x_2$ is reset. By Lemma 3, the total number of different ways in which $x_2$ can be reset before reaching $(q_1, q_2)$ and resetting $x_1$ is bounded by a polynomial $p$ in $|\mathcal{A}_1| + |\mathcal{A}_2|$. Hence the total number of times a combined state $(q_1, q_2)$ can be reached by $\tau$ directly after resetting $x_1$ is bounded by $|Q_1| * |Q_2| * p(|\mathcal{A}_1| + |\mathcal{A}_2|)$. This is polynomial in $|\mathcal{A}_1|$ and $|\mathcal{A}_2|$. By symmetry, this also holds for combined states that are reached directly after resetting $x_2$. Hence, the total numer of resets of either $x_1$ or $x_2$ by $\tau$ is bounded by a polynomial in $|\mathcal{A}_1| + |\mathcal{A}_2|$. Since, by Proposition 4, the length of $\tau$ is bounded by this number, 1-DTAs are polynomially distinguishable.

## 6 Discussion and Conclusions

In this paper we have shown that deterministic timed automata (DTAs) cannot be identified efficiently in the limit (Theorem 1). Moreover, this even holds if the class of DTAs is only allowed access to two clocks (Corollary 1). Furthermore, we have shown that DTAs with a single clock (1-DTAs) are polynomially distinguishable (see Definition 6 and Theorem 5). Polynomial distinguishability is a necessary condition for efficient identifiability in the limit (Lemma 1). Therefore, it is an important step for proving the efficient identifiability of 1-DTAs.

It is possible to construct for every DTA a DFA that accepts the same language. However, this DFA is exponentially larger than the original DTA. Therefore, it is not unexpected that DTAs cannot be identified efficiently. For 1-DTAs, the standard method of creating a DFA that accepts the same language (i.e., the region construction [1]) still results in an exponential blowup of the amount of states. Therefore, one may guess that 1-DTAs can not be identified efficiently. Surprisingly, however, in this paper we have shown that 1-DTAs are polynomially distinguishable, which makes them very likely to be efficiently identifiable.

Currently, we are writing an algorithm that we intend to use to prove efficient identifiability based on the results in this paper. The idea is to write a state merging and transition splitting algorithm like our algorithm for identifying simple TAs (see [5]) that uses polynomial distinguishing strings to ensure the correct identification of 1-DTAs. This is similar to the way state merging was used to show the efficient identifiability of DFAs (see [12]).

Besides allowing us to write such an algorithm, the results in this paper have several other important consequences and/or possible applications. We now give a few examples of consequences and applications.

The fact that 1-DTAs are polynomially distinguishable relies on an important lemma regarding their modeling power (Lemma 2). We believe this lemma has consequences beyond the scope of the 1-DTA identification problem. For example, when model checking a system of two 1-DTAs, the search space may be reduced by restricting the search to smaller valuations in combined states.

The efficiency results have important consequences for anyone interested in identifying timed systems (and TAs in particular). Most importantly, they tell us that 1-DTAs seem to be a good model for identifying a timed system. Furthermore, they show that anyone who needs to identify a DTA with two or more clocks should either be satisfied with sometimes requiring an exponential amount of data, or he or she has to find some other method to deal with this problem. This also holds for other learning frameworks.

For instance, in related work, a query learning algorithm is described for identifying event recording automata (ERAs) [8]. An important property of ERAs is that they are determinizable [7]. This property ensures that the language inclusion problem is decidable for determinizable TAs. Since language inclusion is relevant for identification, this seems to indicate that ERAs are a class of automata that are well-suited for identification. However, a class of automata can only be identified efficiently from queries if it is also efficiently identifiable in the limit from data [14]. Thus, our results show that ERAs can never be identified efficiently since an ERA has access to multiple clocks. We believe it would be interesting, and very valuable for real-world applications, to adapt the timed query learning algorithm to the class of 1-DTAs. Our results indicate that this may result in an efficient query learning algorithm for timed systems.

*Acknowledgement.* We would like to thank Pieter Adriaans for the helpful comments on an early draft of this paper. This research has been supported and funded by the Dutch Ministry of Economic Affairs under the SENTER program, project IS041022 Real-Time Optimalisatie Motor Management.

## References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Larsen, K.G., Petterson, P., Yi, W.: Uppaal in a nutschell. International journal on software tools for technology transfer 1(1-2), 134–152 (1997)
3. Sipser, M.: Introduction to the Theory of Computation. PWS Publishing (1997)
4. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77 (1989)
5. Verwer, S., de Weerdt, M., Witteveen, C.: An algorithm for learning real-time automata. In: Benelearn, pp. 128–135 (2007)
6. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433. Springer, Heidelberg (1998)
7. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. Theoretical Computer Science 211(1), 253–273 (1999)
8. Grinchtein, O., Jonsson, B., Petterson, P.: Inference of event-recording automata using timed decision trees. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 435–449. Springer, Heidelberg (2006)
9. Gold, E.M.: Complexity of automaton identification from given data. Information and Control 37(3), 302–320 (1978)
10. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the ACM 40(1), 95–142 (1993)
11. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
12. Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: Pattern Recognition and Image Analysis. Series in Machine Perception and Artificial Intelligence, vol. 1, pp. 49–61. World Scientific, Singapore (1992)
13. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27 (1997)
14. Parekh, R., Hanovar, V.G.: On the relationship between models for learning in helpful environments. In: Oliveira, A.L. (ed.) ICGI 2000. LNCS (LNAI), vol. 1891, pp. 207–220. Springer, Heidelberg (2000)

# Evaluation and Comparison of Inferred Regular Grammars[*]

Neil Walkinshaw, Kirill Bogdanov, and Ken Johnson

The Department of Computer Science, The University of Sheffield, Regent Court, 211 Portobello, S1 4DP Sheffield, U.K.
{n.walkinshaw,k.bogdanov,csken}@dcs.shef.ac.uk

**Abstract.** The accuracy of an inferred grammar is commonly computed by measuring the percentage of sequences that are correctly classified from a random sample of sequences produced by the target grammar. This approach is problematic because (a) it is unlikely that a random sample of sequences will adequately test the grammar and (b) the use of a single probability value provides little insight into the extent to which a grammar is (in-)accurate. This paper addresses these two problems by proposing the use of established model-based testing techniques from the field of software engineering to systematically generate test sets, along with the use of the Precision and Recall measure from the field of information retrieval to concisely represent the accuracy of the inferred machine.

**Keywords:** Evaluation, State-Merging, Model-Based Testing, Precision and Recall, FSM Testing.

## 1 Introduction

Inferring an unknown regular grammar from a sample of valid and invalid sentences is a well-established problem [13]. In practice it is often difficult to collect a representative sample of valid and invalid sentences that suitably encapsulates every required behavior of the target grammar. This has spurred the development of inductive approaches [19,16], which can produce a reasonable inferred grammar even if the provided set of sentences is only a sparse sample.

The ability to reliably measure the accuracy of an inferred grammar is fundamental to the evaluation of a technique as a whole. Conventionally, the accuracy of a grammar is evaluated by generating a random 'test' sample from the target grammar, and counting the proportion of tests that are correctly classified by the inferred grammar.

In this paper we point out that the standard evaluation techniques of regular grammars can present a skewed perspective of their accuracy. We outline the two main challenges and propose the use of software engineering and information retrieval methods to tackle them.

[*] This work has been funded by the AutoAbstract EPSRC grant EP/C511883/1.

**1. Generating a representative test sample:** Certain aspects of the grammar (a) can be difficult to exercise with random tests, but (b) represent key language features. To address this problem we propose the use of test-generation techniques from the field of model-based testing [17] in software-engineering. Many conformance testing algorithms have been designed to establish with certainty if an implementation of a software system conforms to a known specification of the system. Both implementation and specification are modeled by deterministic finite automata, and so it becomes straightforward to apply these techniques for the sake of establishing the accuracy of inferred grammars.
**2. Measuring the accuracy of the inferred grammar:** Conventionally, an inferred grammar is evaluated by measuring the proportion of correctly classified sentences. However, this provides little insight into grammar properties that are of interest such as exactness or completeness.

Instead of the traditional single value to quantify machine accuracy, this paper illustrates the use of precision and recall [24], a well understood measure from the field of information retrieval. This can be visualised in an accessible manner, and can be used to establish to what extent the approach over / under generalises. We show how this correlates with Dupont's lattice-based representation [10] of the regular inference search space.

The format of the paper is as follows. Section 2 introduces the regular inference problem, and shows how inferred grammars are conventionally evaluated. Section 3 describes state machine testing techniques. Precision and recall are described in Section 4 with their relation to an established lattice-based inference search space shown in Section 5. Section 6 contains a small case study. Related work and conclusions can be found in Sections 7 and 8 respectively.

## 2 Regular Inference and the Evaluation of Inferred Grammars

This section provides a brief background to the challenge of regular inference, followed by a description of the conventional evaluation approach along with its pitfalls. First, we list some basic definitions and set our notation based on Dupont *et al.* [9,10]. A deterministic finite automaton (DFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ is a finite alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a partial function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accepting states. We say a sequence $s \in \Sigma^*$ is *accepted* by $A$ if there is a path from the initial state $q_0$ to an accepting state $q$. In symbols, $q_0 \xrightarrow{s} q$, for $q \in F$. Throughout this paper, we only consider a special type of DFA where all states are accepting states. A labelled transition system (LTS) $A$ is a DFA whenever $F = Q$. The consequence of this is that for any sequence accepted by an LTS, all prefixes of that sequence are also accepted. That is, every LTS is *prefix closed*.

A *grammar* is a set of rules which specify a subset $L$, called a *language*, from the set $\Sigma^*$ of all possible sequences of characters in $\Sigma$. We use deterministic finite automata to represent regular grammars and write $A(L)$ to denote a DFA $A$ which produces the language $L$. When referring to two or more languages, we denote the language produced by the DFA $A$ as $L_A$; when no ambiguities arise, the subscript is omitted. We define the concatenation $L_A L_{A'}$ of two languages $L_A$ and $L_{A'}$ to be the set $\{ls \mid l \in L_A,\ s \in L_{A'}\}$. For a state $q \in Q$ define $L_A(q) = \{s \in \Sigma^* \mid q \xrightarrow{s} q_f$ for some $q_f \in Q\}$ to be the language accepted by $A$ in $q$.

Given sample sets of *valid* sequences which are in $L$ and optionally *invalid* sequences that are not in $L$, the *regular grammar inference problem* is to identify a regular grammar which defines $L$, using the samples. That is, given $S^+ \subseteq L$ and $S^-$ such that $S^- \cap L = \emptyset$, construct a DFA $A(L) = (Q, \Sigma, \delta, q_0, F)$ where for each $l \in L$ there exists a $q \in Q$ such that $q_0 \xrightarrow{l} q$, and for all $s \in S^-$ there is no such a path.

The DFA $A$ is a prefix tree acceptor (PTA) [20,22] of $S^+$ if each sequence in $S^+$ has a unique path from the start state to an accepting state, with common prefixes sharing the same path. If a set $S^-$ is available, it is trivial to augment the PTA to include them, preventing false merges (below) from occurring. This is conventionally referred to as the *augmented prefix tree acceptor* (APTA).

*State merging* techniques take a PTA or APTA as input, and proceed to merge states that are deemed to be equivalent. Ultimately they aim to converge on the most general machine that is consistent with the given samples. We recall a partition $\pi$ maps a set $Q$ to a disjoint family of subsets whose union is $Q$. For $q \in Q$, $\pi(q)$ is the unique subset containing $q$; when states are merged, they are placed in the same subset $\pi(q)$. Let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA. A *quotient automaton* [10] $A/\pi = (Q_{A/\pi}, \Sigma, \delta_{A/\pi}, q_{A/\pi}, F_{A/\pi})$ is derived from $A$ with respect to a partition $\pi$ as follows. The set of states $Q_{A/\pi}$ is defined as $Q_{A/\pi} = \{\pi(q) \mid q \in Q_A\}$ and the set of final states $F_{A/\pi}$ as $F_{A/\pi} = \{\pi(q) \mid q \in F_A\}$. Let $Q, Q' \in \pi(Q_A)$ and $a \in \Sigma$. Define the transition function $\delta_{A/\pi} : Q_{A/\pi} \times \Sigma \to Q_{A/\pi}$ as $\delta_{A/\pi}(Q, a) = Q'$ if, and only if, there exists $q \in Q$ and $q' \in Q'$ such that $\delta_A(q, a) = q'$, for $a \in \Sigma$. The initial state $q_{A/\pi}$ is the set $q_{A/\pi} = \pi(q_A)$. As a consequence of state merging, any path that is accepted in $A$ is also accepted in $A/\pi$. In fact, $A/\pi$ is a generalization of $A$ where $L_A \subseteq L_{A/\pi}$.

Given a DFA $A$, the set $S^+$ is *structurally complete* with respect to $A$ if it covers every transition in $A$. If one uses a PTA built from such an $S^+$ and there is sufficient information to prevent an incorrect merge from occurring, the state merging process can be guaranteed to correctly converge at $A(L)$. In other words, then there exists a partitioning $\pi$ such that the quotient automaton $PTA/\pi = A(L)$ [10]. This requirement is however unrealistic for most real-world applications; if the target machine is unknown it is often difficult to guarantee structural completeness, and if the target automaton is nontrivial, a complete set of samples can simply be too large to obtain or difficult to identify. In practice, techniques need to be able to infer fairly accurate grammars given only sparse samples. Spurred on by several competitions, a number of promising state-merging techniques have emerged [16,9]. The ability of these approaches to infer reasonably accurate grammars from sparse samples, coupled with the scalability of these techniques, renders them particularly appealing for many practical applications.

Grammar inference techniques are evaluated in terms of their accuracy at classifying a test sample of sequences [15,16,4,9]. The set of test sequences (referred to as a *test set*) is usually compiled by tracing a selection of random paths over the target machine, ensuring that they are evenly split between sequences that should be rejected and accepted, as well as ensuring that their lengths fit a uniform distribution. The accuracy is then measured as the proportion of test sequences that are correctly classified as either accepted or rejected by the inferred grammar.

This approach to evaluation is however problematic for two reasons. The validity of the accuracy metric is entirely dependent upon a test set that is representative of the target machine, and this is often improbable if the test set generation process is essentially random. The second reason is that, even if a representative test set is found, a single value provides very few insights into what might make one approach to grammar inference superior / inferior to another. These two problems are elaborated below.

Obtaining a test set that is 'representative' of the target grammar is very challenging. Usually, the language of the target machine will contain an infinite number of possible valid (and invalid) sequences, but it is not sufficient to simply pick a random sample for the sake of testing. Grammars that correspond to a large DFA with a large alphabet have a low observability: certain aspects are much less likely to appear in a random sequence than others. Bongard and Lipson [4] use the example of the Tomita 1 language [21], that only produces a positive classification for a binary string 2.4% of the time. As demonstrated in Lang's experiments with random DFAs [15], the size of a random test set that approximately infers a grammar invariably has to increase by orders of magnitude as the size of the target machine increases.

Alongside the problem of identifying a suitable test set, there is also the problem of how to interpret the final accuracy result - a single value provides very few qualitative insights into the resulting grammar. Assuming, for example, that we can confidently assert that a DFA has an accuracy of 70%. Does this mean that it is more likely to falsely classify a string that should be accepted or rejected? If we wanted to improve its accuracy, would we need to make it more general or more specific?

## 3 Model-Based Test Generation

The challenge of identifying a test set that reliably covers every behavior of some specification DFA is nontrivial. Random sets of strings may easily explore specific aspects of the machine much more thoroughly than others, and hence may result in a skewed accuracy measurement. How do we identify a finite set of strings that can be used to evaluate the accuracy of the hypothesis machine?

This problem has been considered in the area of model-based software testing where the focus is to check by testing whether an implementation is similar to a model. Testing uses a model (reflecting the intended behavior) and generates test sequences from it. If an implementation produces a different result to a model on any of those sequences, such an implementation is considered faulty; otherwise, one may wish to have a confidence that an implementation is similar to a model. The extent of such a similarity depends on the specific testing method used to generate test sequences and on the properties of both a model and an implementation. In the context of this paper, the aim is to compare two DFAs, the hypothesis and the target. For this reason, the focus is on testing methods which can demonstrate an equivalence between languages accepted by the two DFAs rather than, for instance, whether one language contains another one.

The problem of checking whether two DFAs are accepting the same language by experiment cannot be solved in the most general case — since a test set has to be finite, it is always possible for an implementation DFA to contain more states than could be explored with a chosen test set and those extra states may have undesired behavior. For this reason, all DFA testing methods assume that it is possible to estimate the maximal number of states in an implementation, in advance. In addition, an alphabet of an implementation is usually assumed to be known. Finally, both machines are expected to be deterministic, minimal and feature a reliable reset (a special input which brings them to their respective initial states, not usually shown on a transition diagram); this simplifies testing and holds in the case considered in this paper. State-based testing methods systematically explore the (unknown) transition structure of an implementation DFA, comparing it to the model. From every state which is included in a model, they attempt every symbol in an alphabet. This verifies that all transitions absent in a model are also absent in an implementation; symbols which label transitions in a model are followed with specific sequences to verify target states of those transitions. This way, every state and every transition in a model are checked in an implementation. Target state verification is based on an assumption that all states in a model are different, i.e. they accept different languages. For this reason, for every pair of states, it is possible to choose a sequence which distinguishes between them and a set of such sequences (called a *characterisation set* below) can be used to check that an implementation has reached the expected states. The testing method used in this paper is the implementation by the authors of the original Vasilevski/Chow W-Method [6].

Construction of a test set using the W-Method is briefly described following [3]. For an implementation DFA $A = (Q, \Sigma, \delta, q_0, F)$ and a model DFA $S$, the W-Method constructs $Y \subseteq \Sigma^*$ such that $(L_A \cap Y = L_S \cap Y) \Rightarrow L_A = L_S$. Such a set is called a *test set* of $A$. A *state cover* $C$ is a prefix-closed subset of $\Sigma^*$ containing all sequences of inputs needed to visit every state of a DFA from the initial state; in symbols, $\epsilon \in C$ and for all states $q \in Q \setminus \{q_0\}$ there exists a $c \in C$ such that $\delta(q_0, c) = q$. For a subset $W \subseteq \Sigma^*$ the states $q_1, q_2 \in Q$ are called *$W$-distinguishable* if $(L_A(q_1) \cap W) \neq (L_A(q_2) \cap W)$. We call $W$ a *characterisation set* [12] of $A$ if any two distinct states of $A$ are $W$-distinguishable. Given an estimate $k$ as to how many more states an implementation may have compared to a model, a test set $Y = C(\{\epsilon\} \cup \Sigma \cup \cdots \cup \Sigma^{k+1})W$.

There is a clear overlap between the two fields of conformance testing and grammar (DFA) inference, because tests generated from a model can be interpreted as membership queries to be answered by a hidden implementation machine to establish whether the two are equivalent. Several software engineering researchers have previously explored the relationship between the two areas [2,14], and the use of conformance testing algorithms for answering equivalence queries posed by Angluin's $L^*$ algorithm.

## 4 Evaluating Accuracy with Precision and Recall

Grammar inference techniques, and classifiers in general, are commonly evaluated in terms of the probability that they will return a correct response [16,9,4]. This measure is often suitable as a coarse summary of classifier behavior, but provides a user with very little insight into any particular strengths / weaknesses of the technique. A single accuracy figure can give no insight into questions such as (a) whether a hypothesis machine over/under generalised and (b) whether a hypothesis language contains too many false positives or negatives.

### 4.1 Precision and Recall in Grammar Inference

Precision and recall [24] is a more descriptive measure, because it quantifies the similarity of two objects with two variables instead of one – precision (exactness) and recall (completeness). Originally from the domain of information retrieval, it is used to measure the overlap between what has been retrieved and what is relevant.

The conventional precision-recall evaluation process works by trying to establish the "overlap" between an inferred model and its target. This is achieved by computing random samples from the inferred and target models, and adding sequences to the $RET$ and $REL$ sets depending on how they are classified. This classification is illustrated in the table below; if a sequence (from either machine) is accepted by both machines, it is added to both $RET$ and $REL$ sets, if the string is accepted only by the inferred machine, then it is added to $RET$ etc. The final RET and $REL$ sets are then used to compute precision and recall as follows: precision is computed by $\frac{|REL \cap RET|}{|RET|}$ and recall by $\frac{|REL \cap RET|}{|REL|}$. Precision and recall have been used in the past to measure the accuracy of inferred models in both the domains of software engineering and grammar inference. As an example, in software engineering, Lo *et al.* [18] use it to establish the accuracy of reverse-engineered software specifications. In grammar inference, Tu and Hanovar [23] use it to measure the accuracy of their context-free grammar inference technique. The two variables reflect complementary aspects of the inferred model, and are more descriptive as a result, helping to answer the above questions that arise with the use of a single "accuracy" measure.

| $H$ Machine (Hypothesis) | $S$ Machine (Specification) | $RET$ | $REL$ |
|---|---|---|---|
| accept | accept | $\times$ | $\times$ |
| accept | reject | $\times$ | |
| reject | accept | | $\times$ |
| reject | reject | | |

Unfortunately, obtaining reliable precision and recall scores when comparing DFAs is not straightforward. The conventional approach has two flaws that can undermine confidence in the results:

**1. Sampling:** It relies on the assumption that the random positive samples computed for each of the machines are thorough enough to capture the differences between the two machines. This approach will only identify disagreements between the machines that are easy to reach with random sequences, and will ignore those that are less likely to be exercised. There is also the danger that the sample will simply represent the training sample, which is often also a random sample from the target machine. Ultimately, any measure of accuracy that is computed from samples that are constructed in this way is at best indicative of the correspondence between two machines, and risks being misleading.
**2. Measuring:** The computation of RET and REL sets (as shown in the table above) is biased towards the accepting behavior of the two machines. As noted above, conventional samples do not include invalid sequences, but even if they did, they would not be accounted for according to the scheme in the table above. If both machines correctly reject a sequence, this would not be incorporated into the computation of precision and recall. In the context of grammar inference, it is as important to evaluate an inferred grammar in terms of the sequences it rejects as well as the sequences it accepts. Hence, approaches that do not use precision and recall (using the conventional single-valued approach) [9] tend to ensure that test samples are evenly split into sets of valid and invalid sequences.

A naive solution to the two problems would be to (a) evenly construct a sample from both valid and invalid sequences, and (b) add invalid sequences to both *RET* and *REL* when the two machines are in agreement about their rejection. This will however still bias the results, because the sampling emphasises those parts of the machine that are easy to reach. There is also the problem that the split between valid and invalid sequences is in effect arbitrary; an even split is making the unlikely assumption that the language of the machine is evenly balanced in terms of its valid and invalid sequences.

### 4.2 Authoritative Measurement of Precision and Recall by Conformance Testing

One apparent solution to the sampling problem mentioned above is to apply conformance testing techniques (see Section 3). For a given DFA, techniques such as the W-Method will produce a finite set of sequences that is not overly biased towards any particular part of the machine. The conventional approach discussed above generates a composite set of random samples from both the inferred and the target machine, in the hope that this will highlight the differences between the two machines. Instead, using model-based testing techniques, it is only necessary to generate one test set from the target machine. This is guaranteed to highlight every discrepency between the two machines.

Although test sets that are generated by techniques such as the W-Method are comprehensive, they do have two characteristics that should be noted when applied for the sake of evaluating inferred grammars,

**1. Scale:** The test set is usually very large. With software and hardware systems, depending on the latency of the system under test, each test execution incurs a cost that can in practice render the execution of a complete test set infeasible. Large test sets are much less of an issue when they are used to evaluate the accuracy of hypothesis grammars. The time taken by test execution is reduced by many orders of magnitude if it merely consists of traversing a path in a graph that is stored in memory. For this reason, this work does not use more efficient testing methods such as Wp and HSI which use a subset of distinguishing sequences depending on a state to be checked (although these methods could just as well be applied).
**2. Partial inclusion of training set:** The test set invariably incorporates a proportion of the training set, which could be seen to bias the results. A certain overlap between the training and test set is inevitable – the prefixes of invalid sequences are valid ones, many of which are a necessary part of a rigorous test set. This overlap can be problematic when the training and test set are both sampled from the same distribution of sequences in the target language; in this context test sequences are meant to test how well the classifier generalises from the training set, and this does not happen if training sequences appear in the test set. However, by generating the test set with a test set generation algorithm, the test set is no longer testing the generalisation of the classifier, but instead serves to produce an absolute measure of difference between the inferred and the target grammars.

Although conformance testing solves the sampling problem, there still remains the challenge of using these tests to accurately measure precision and recall. As mentioned earlier, the conventional approach used to work out the values of $RET$ and $REL$ relies on the assumption that the set of samples contains an equal number of valid and invalid sequences. However, in the case of the W-Method, the vast majority of the tests test for invalid behavior (making sure that the machine does not have extra transitions). This is a property of models of software where an alphabet is a set of commands and from each state only a small subset of those commands can be executed. Numerous invalid sequences can result in skewed precision and recall results: even if the inferred machine does not accept any of the sequences it should, if it correctly rejects most of the invalid sequences, the precision and recall will be disproportionately high.

This problem is addressed by refining the conventional scoring approach to distinguish between accepting and rejecting behavior. Instead of computing a single precision and recall pair, we compute one that describes the accuracy of the hypothesis machine $H$ in terms of the set of traces it should accept, and the other in terms of the set of traces it should reject. For this reason, we divide $RET$ and $REL$ into $RET^+$, $RET^-$, $REL^+$ and $REL^-$. Test sequences can thus be categorised according to the table below.

| $H$ Machine (Hypothesis) | $S$ Machine (Specification) | $RET^+$ | $REL^+$ | $RET^-$ | $REL^-$ |
|---|---|---|---|---|---|
| accept | accept | × | × | | |
| accept | reject | × | | | × |
| reject | accept | | × | × | |
| reject | reject | | | × | × |

If a sequence is accepted by $H$ but should in fact be rejected, it is added to $RET^+$ and $REL^-$ as shown in Row 2. Thus, $precision^+ = \frac{|REL^+ \cap RET^+|}{|RET^+|}$ and $recall^+ = \frac{|REL^+ \cap RET^+|}{|REL^+|}$, and the same approach is used to compute the negative precision and recall from $RET^-$ and $REL^-$. The above definitions of positive and negative precision and recall can be interpreted as follows: $precision^+$: high value means that the positive sequences represented by the hypothesis machine are largely correct; $recall^+$: high value means that the set of positive sequences represented by the hypothesis machine is largely complete; $precision^-$: high value means that the negative sequences represented by the hypothesis machine are largely correct; $recall^-$: high value means that the set of negative sequences represented by the hypothesis machine is largely complete.

## 5 Relationship between Precision, Recall and the State-Merging Search Space

Precision and recall are more suitable for characterising the success of state merging sequences than the traditional single-valued accuracy approach. By adopting the precision and recall metrics introduced in Section 4.1, it is possible to determine with greater certainty whether a particular merge improves machine accuracy or not. This is best illustrated with the lattice-based characterisation of the state-merging search space.

Inference techniques invariably involve searching a potentially very large space of hypotheses in order to arrive at some result. The representation of this space is key to their efficiency. A representation might, for example, indicate that the selection of one hypothesis would rule out the subsequent selection of another (potentially more suitable) hypothesis. This sort of information can be very valuable during the inference process.

In regular inference hypotheses are commonly related to each other in terms of their respective generality [1]. A grammar $B$ is more general than $A$ if $B$ is the quotient of a merge (Section 2) in $A$. In symbols, let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA and $P(A)$ denote the set of all partitions of the state space $Q_A$ of $A$. We define a partial order on $P(A)$, as described in [10]: let $\pi_1 \in P(A)$ such that $\pi_1(Q_A) = \{Q_1, \ldots, Q_r\}$. Define $\pi_2 = \{Q_j \cup Q_k\} \cup \pi_1 / \{Q_j, Q_k\}$, $1 \leq j, k \leq r, j \neq k$. We say that $\pi_2$ *derives* from $\pi_1$, denoted $\pi_2 \geq \pi_1$. This derivation operation on partitions defines a partial ordering on the set $P(A)$.

A useful metric for the evaluation of a hypothesis machine should also be useful as a guide for the search process. In terms of the lattice search-space

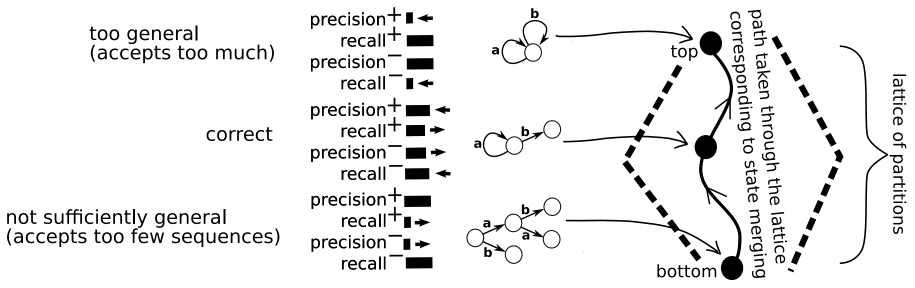


**Fig. 1.** Lattice-based search space

(Figure 1), it should be possible to tell whether a given solution is too general or too specific, to direct the search. From this perspective the traditional single-valued accuracy score can be highly misleading. For example, given a test set where half of the sequences should be valid, and half invalid, the universal DFA at the top of the lattice would result in a score of about 50% (despite the fact that the machine is grossly overgeneralised). Conversely, it is possible that the PTA at the bottom of the lattice produces a similar score, by correctly rejecting most of the negative tests, but also incorrectly rejecting most of the positive tests. During the merging process, the score could fluctuate, but not necessarily provide any guidance — there is thus no relationship between a path through the search-space, and a definite increase or decrease in accuracy.

When adopting precision and recall, there is a more direct relationship with the search-space. In the figure above we depict values of precision and recall using bars of different width. The structurally complete PTA at the bottom of the figure is the starting point of the merging process. In the course of merging states (reflected by a path through the lattice of partitions of states of the initial PTA) one aims to increase $Recall^+$ and $Precision^-$ without compromising $Precision^+$ and $Recall^-$. If the process of generalisation goes too far, the overly-general outcome has low positive recall and negative precision, but the high positive precision and negative recall.

For any pair of hypothesis $\pi_A$, $\pi_B$ in the lattice where $\pi_B \geq \pi_A$, we can state the following: $Precision^+(\pi_B) \leq Precision^+(\pi_A)$, $Recall^+(\pi_B) \geq Recall^+(\pi_A)$, $Precision^-(\pi_B) \geq Precision^-(\pi_A)$, and $Recall^-(\pi_B) \leq Recall^-(\pi_A)$.

## 6 Case Study: Effect of DFA Generalisation on Accuracy

In this section we use a small case study to illustrate how the precision and recall measures, paired with a model-based test set generation strategy can provide more accurate and detailed quantitative insights into the performance of inference algorithms. The purpose of this case study is to illustrate the utility of our proposed evaluation technique (as opposed to the efficacy of any specific state-merging algorithms). Although the case study subject is quite specific, it is probable that the extra insights garnered from combining precision and recall with model-based test sets would apply to most evaluations of regular inference techniques.

As a basis for the case study, we compare the performance of two variants of the well-known EDSM merging algorithm [16] at inferring a randomly generated grammar. The EDSM algorithm selects suitable state merges by assigning a score for every pair of states (we use the Blue-Fringe algorithm [16] to select these pairs). The first variant merges states where the score $\geq 1$, and the second one merges states where the score $\geq 2$. The target grammar is randomly generated; its DFA has 50 states, and has an alphabet of 100[1].

The training set is generated in the traditional way by tracing a selection of random paths across the target machine. We chose a small sample size to emphasise the performance of the algorithms with respect to sparse samples. The sample is composed of 50 valid and 50 invalid sequences. The length of each path is a random number between 2 and $n + 5$, where $n$ is the diameter of the target DFA. For each variant of the inference algorithm, the accuracy was charted for every iteration (state-merge), using both the traditional single-valued measurement, and the precision and recall approach proposed in this paper. The results are shown in Figure 2, with the left chart showing results for the case of merge threshold $\geq 1$, middle chart showing the results for merge threshold $\geq 2$. The single-value accuracy (black line) was established by using a test set consisting of random traces over the target grammar, whereas the test set for the precision and recall scores was generated using the W-Method.

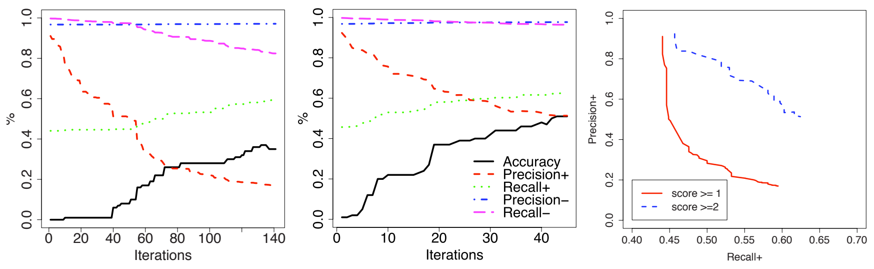


**Fig. 2.** Traditional Accuracy vs. Precision-Recall measurements at each iteration of a state merging algorithm, for two variants of the EDSM algorithm

In both charts, the single-valued accuracy score has a similar shape. It starts at zero, and finshes at a score of about 40-50% accuracy. With a higher merge threshold it is slightly steeper, suggesting that it produces more accurate results early-on in the state-merging process. In isolation, the single-valued accuracy line in both charts suggests that the state merging process is moving towards

[1] A GraphML file containing the target DFA can be downloaded from `http://www.dcs.shef.ac.uk/~nw/Files/icgiExample.xml` The source code for the W-Method and comparison of two DFAs is part of a larger framework developed by the authors, available from `http://statechum.sourceforge.net/`.

an increasingly accurate result. This is however somewhat deceptive. Looking at the precision and recall scores, it becomes apparent that, although the positive recall is increasing, the positive precision is substantially reduced as the merging continues. This is clearest in the left chart around the 58th iteration, a merge happens that reduces the positive precision by about 10%, whereas at the same point the accuracy score increases sharply.

For the sake of illustration, the purpose of this study is to investigate the performance difference in the EDSM algorithm for different thresholds. The single-valued measure gives little insight in this respect. The precision and recall scores on the other hand are much more descriptive. They show that the difference in accuracy between the two versions is due to a combined improvement in positive precision and negative recall. In practice this means that with a lower merging threshold the resulting DFA accepts too many false positives and negatives. On the left chart, the positive precision drops by about 70% throughout the inference process, whereas in the middle chart it only drops by about 40%. The negative recall drops by about 20% in the left chart, it stays around 95% in the second one.

Finally, precision and recall can be visualised together, to provide an overall summary of the accuracy of a particular search technique. For the sake of simplicity, we omit the rejecting behavior in this case, where there is only a small trade off between precision and recall. The rightmost chart in Figure 2 plots the positive precision versus the positive recall for the two EDSM variants, as measured using the W-Method test set. These plots [24] clearly depict the performance increase for the higher score threshold, where the precision is compromised to a much lesser extent as the recall increases.

## 7 Related Work

The problem of measuring the accuracy of learner hypotheses forms the basis of a substantial amount of discussion in machine learning literature. Receiver Operator Characteristic (ROC) curves have emerged as a useful solution to this problem [5]. These make explicit the relationship between the number of true-positives and false-positives and are closely related to precision and recall. However, if the data set upon which the curve is built is skewed (which is usually the case with grammar inference data sets), precision and recall is a preferable measure [7]. To the best of the authors' knowledge, precision and recall have not been applied in the context of regular grammar inference.

Competitions, most notably the Abbadingo competition [16], have played a major role in driving the development of new inference techniques. These are usually operated by setting up a server that randomly generates a (hidden) target DFA, along with an accompanying random training and test set. Conventionally, a winning inference technique has to be able to produce a hypothesis DFA that produces an accuracy score of 99%. As we have shown, depending on the test set and the target machine, this accuracy score can be misleading. In the context of such a competition, this has the potential to result in the selection of inference techniques that might not fare as well if the criterion for success was that the resulting machine should produce high precision and recall scores, and if the test set was generated systematically as opposed to randomly.

The lattice-based representation of search-space (and its relation with precision and recall) is particularly relevant to heuristic grammar inference techniques that depend upon a notion of "fitness". As an example, Dupont's GIG method [8] uses a genetic algorithm to search the lattice of automata, where one search result is considered to be "fitter" than another if it has fewer states and misclassifies fewer strings in $S^-$. If we assume that part of the (unused) training sample can be used as a more complete test set, then it becomes possible to work out the positive and negative precision and recall for each automaton. This measure could then be used as a more fine-grained fitness function and could form a suitable basis for a multi-objective search-based inference algorithm [11].

## 8 Conclusions

Due to the fact that most target machines are inherently unbalanced, the conventional use of evenly-split random traces in the target machine as test sets is insufficient, and can provide a skewed view of the accuracy of the final machine. The use of a single value to summarise this accuracy is too simplistic, and does not provide enough of an insight into why a particular inference algorithm becomes inaccurate. In this paper we have shown how the use of precision and recall, combined with a systematic test set generation strategy, can be used to evaluate inferred grammars in an authoritative manner.

We distinguish between the precision and recall for rejecting and accepting behavior of the inferred machine because the target machine is usually unbalanced. This provides a more detailed means for evaluating machines and makes it easier to see whether a hypothesis machine has been under or over generalised. We have also shown how this means of evaluation links in with the established lattice-based view of the inference search space.

## References

1. Angluin, D., Smith, C.H.: Inductive inference: Theory and methods. Computing Surveys 15(3), 237–269 (1983)
2. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005)
3. Bogdanov, K., Holcombe, M., Ipate, F., Seed, L., Vanak, S.: Testing methods for X-Machines: A review. Formal Aspects of Computer Science 18, 3–30 (2006)
4. Bongard, J., Lipson, H.: Active coevolutionary learning of deterministic finite automata. Journal of Machine Learning Research 6, 1651–1678 (2005)
5. Bradley, A.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 30(7), 1145–1159 (1997)
6. Chow, T.: Testing Software Design Modelled by Finite State Machines. IEEE Transactions on Software Engineering 4(3), 178–187 (1978)

7. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: ICML. ACM International Conference Proceeding Series, vol. 148, pp. 233–240. ACM, New York (2006)
8. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 236–245. Springer, Heidelberg (1994)
9. Dupont, P., Lambeau, B., Damas, C., van Lamsweerde, A.: The QSM algorithm and its application to software behavior model induction. Applied Artificial Intelligence 22, 77–115 (2008)
10. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 25–37. Springer, Heidelberg (1994)
11. Fonseca, C., Fleming, P.: An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation 3(1), 1–16 (1995)
12. Gill, A.: Introduction to the Theory of Finite State Machines. McGraw-Hill, New York (1962)
13. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
14. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. Logic Journal of the IGPL 14(5), 729–744 (2006)
15. Lang, K.: Random DFA's can be approximately learned from sparse uniform examples. In: COLT, pp. 45–52 (1992)
16. Lang, K., Pearlmutter, B., Price, R.: Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 1–12. Springer, Heidelberg (1998)
17. Lee, D., Yannakakis, M.: Principles and Methods of Testing Finite State Machines - A Survey. Proceedings of the IEEE 84, 1090–1126 (1996)
18. Lo, D., Khoo, S.: QUARK: Empirical assessment of automaton-based specification miners. In: WCRE, pp. 51–60. IEEE Computer Society, Los Alamitos (2006)
19. Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: Pérez de la Blanca, N., Sanfeliu, A., Vidal, E. (eds.) Pattern Recognition and Image Analysis. Series in Machine Perception and Artificial Intelligence, vol. 1, pp. 49–61. World Scientific, Singapore (1992)
20. Parekh, R., Honavar, V.: Grammar Inference, Automata Induction, and Language Acquisition, ch. 29. Marcel Dekker, USA (2000)
21. Tomita, M.: Dynamic construction of finite-state automata from examples using hill-climbing. In: Proceedings of the Fourth Annual Cognitive Science Conference, Ann Arbor, Mi, pp. 105–108 (1982)
22. Trakhtenbrot, B., Barzdin, Y.: Finite Automata, Behavior and Synthesis. North Holland, Amsterdam (1973)
23. Tu, K., Honavar, V.: Unsupervised learning of probabilistic context-free grammar using iterative biclustering. Technical Report 00000572, Dept. Computer Science, Iowa State University (May 2008)
24. van Rijsbergen, C.J.: Information Retrieval. Butterworth-Heineman, Newton (1979)

# Identification in the Limit of $k, l$-Substitutable Context-Free Languages

Ryo Yoshinaka

Graduate School of Information Science and Technology, Hokkaido University, North-14 West-9, Sapporo, Japan
ry@ist.hokudai.ac.jp

**Abstract.** Recently Clark and Eyraud (2005, 2007) have shown that substitutable context-free languages are polynomial-time identifiable in the limit from positive data. Substitutability in context-free languages can be thought of as the analogue of reversibility in regular languages. While reversible languages admit a hierarchy, namely $k$-reversible regular languages for each nonnegative integer $k$, Clark and Eyraud targeted the subclass of context-free languages that corresponds to zero-reversible regular languages only. Following Clark and Eyraud's proposal, this paper introduces a hierarchy of substitutable context-free languages as the analogue of that of $k$-reversible regular languages and shows that each class in the hierarchy is also polynomial-time identifiable in the limit from positive data.

## 1 Introduction

Efficient learning of context-free languages is a topical issue on grammatical inference (see e.g. de la Higuera [12], Lee [18]), but not many techniques are known to be applicable to identification in the limit from positive data of non-regular subclasses of context-free languages, in comparison with subclasses of regular languages (see e.g. Lange et al. [17]). Recently Clark and Eyraud [6,7] have shown that *substitutable context-free languages* are polynomial-time identifiable in the limit from positive data. Their work is remarkable among other achievements on learning context-free languages in several regards. One is the efficiency of the learning algorithm. Their algorithm for substitutable context-free languages runs in time polynomial in the size of the given data and it admits a set of positive examples of polynomial cardinality in the description size of the target grammar on which the conjecture converges to the target language. The second virtue is that the notion of substitutability can explain an aspect of natural language phenomena, which meets the very first motivation of grammatical inference [9]. A language $L$ is said to be substitutable if and only if

$$x_1y_1z_1, x_1y_2z_1, x_2y_1z_2 \in L \text{ implies } x_2y_2z_2 \in L$$

for any strings $x_1, y_1, z_1, x_2, y_2, z_2$. From the point of view of formal language theory, substitutability of context-free languages can be thought of as the exact analogue of *zero-reversibility* in regular languages. Angluin [1] introduced the hierarchy of $k$-reversible languages for nonnegative integers $k$ and showed polynomial-time learnability of $k$-reversible regular languages.[1] A language $L$ is $k$-reversible if and only if

$$x_1 v y_1, x_1 v y_2, x_2 v y_1 \in L \text{ implies } x_2 v y_2 \in L$$

where the length of $v$ is $k$. As the literature has paid much attention to reversible regular languages and their variants and obtained many fruitful results (e.g., [2, 14,16,15,13,19,21,23]), the close relation of substitutable context-free languages to reversible regular languages also seems an advantage of their study. In fact Clark and Eyraud [7] suggested that one may define for context-free languages the exact analogue of $k$-reversibility in regular languages and that such classes would be still polynomial-time identifiable in the limit from positive data. This paper answers to those expectations in the affirmative. We call a language $L$ $k, l$-substitutable if and only if

$$x_1 v y_1 u z_1, x_1 v y_2 u z_1, x_2 v y_1 u z_2 \in L \text{ implies } x_2 v y_2 u z_2 \in L$$

where the length of $v$ is $k$ and that of $u$ is $l$. This paper proves that $k, l$-substitutable context-free languages are identifiable in the limit from positive data by a polynomial-time algorithm that is a natural generalization of Clark and Eyraud's one.

## 2 Definitions

We start by some standard notation, most of which follows Clark and Eyraud [7]. Let $\Sigma$ be a non-empty finite set. $|\Sigma|$ denotes its cardinality. If $x$ is a finite sequence consisting of elements of $\Sigma$, it is called a *string (over $\Sigma$)* and $|x|$ denotes its length. $\lambda$ is the *empty string*. $\Sigma^*$ denotes the set of all strings over $\Sigma$, $\Sigma^+ = \Sigma^* - \{\lambda\}$, $\Sigma^k = \{\, x \in \Sigma^* \mid |x| = k \,\}$, $\Sigma^{\le k} = \{\, x \in \Sigma^* \mid |x| \le k \,\}$ and $\Sigma^{<k} = \Sigma^{\le k} - \Sigma^k$. For $x \in \Sigma^*$ and $a \in \Sigma$, $|x|_a$ denotes the number of occurrences of $a$ in $x$. Any subset of $\Sigma^*$ is called a *language (over $\Sigma$)*. If $L$ is a finite language over $\Sigma$, its size is defined as $\|L\| = \sum_{w \in L} |w|$. We shall assume an order $\prec$ or $\preceq$ on $\Sigma$ which we shall extend to $\Sigma^*$ in the canonical way by saying that $u \prec v$ if either $|u| < |v|$ or $|u| = |v|$ and $u$ is lexicographically before $v$.

A *context-free grammar (* CFG*)* is denoted by a quadruple $G = \langle \Sigma, V, P, S \rangle$, where $\Sigma$ is the finite set of *terminal symbols*, $V$, disjoint from $\Sigma$, is the finite set of *nonterminal symbols*, $P$ is the finite set of *production rules* and $S \in N$ is the *start symbol*. A production rule in $P$ has the form $A \to \beta$ for some $A \in V$ and $\beta \in (\Sigma \cup V)^+$. If $A \to \beta \in P$, we write $\alpha A \gamma \Rightarrow_G \alpha\beta\gamma$ for any $\alpha, \gamma \in (\Sigma \cup V)^*$. $\Rightarrow_G^+$ is the transitive closure of $\Rightarrow_G$ and $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$. The subscript $G$ of $\Rightarrow_G$ is omitted if it is understood from the context. The *context-free language* (CFL) $\mathcal{L}(G)$ generated by $G$ is the set $\mathcal{L}(G, S)$, where $\mathcal{L}(G, \alpha) = \{\, w \in \Sigma^* \mid \alpha \overset{*}{\Rightarrow} w \,\}$ for $\alpha \in (\Sigma \cup V)^*$. Two grammars $G_1$ and

[1] Angluin defined $k$-reversible languages as a subclass of regular languages, while this paper calls any language satisfying $k$-reversibility a $k$-reversible language.

$G_2$ are *equivalent* iff $\mathcal{L}(G_1) = \mathcal{L}(G_2)$. The description size of $G$ is defined as $\|G\| = \sum_{A \to \beta \in P}(|A\beta|)$. A symbol $A \in \Sigma \cup V$ is *useless* in $G$ if there are no $x, y, z \in \Sigma^*$ such that $S \overset{*}{\Rightarrow} xAz \overset{*}{\Rightarrow} xyz$. A CFG $G$ is *reduced* iff every $A \in \Sigma \cup V$ is not useless. We assume all grammars to be reduced in this paper. Note that we do not allow empty right hand side to production rules, and thus any CFLs dealt with in this paper are $\lambda$-free.

In the following, terminal symbols will be indicated by $a, b, c, \dots$, nonterminal symbols by $A, B$, strings over $\Sigma$ by $u, v, \dots, z$, and strings over $(\Sigma \cup V)^*$ by $\alpha, \beta, \gamma, \delta$.

We now define our learning criterion. This is *identification in the limit from text* (or equivalently *from positive data*) as defined by Gold [9]. Let $\mathbb{R}$ be any recursive set of finite descriptions, say CFGs, and $\mathcal{L}$ be a function from $\mathbb{R}$ to non-empty languages over $\Sigma$. A learning algorithm $\mathcal{A}$ on $\mathbb{R}$, is an algorithm that computes a function from finite sequences of strings $w_1, \dots, w_n \in \Sigma^*$ to $\mathbb{R}$. We define a *presentation* of a language $L$ to be an infinite sequence of elements (called *positive examples*) of $L$ such that every element of $L$ occurs at least once. Given a presentation, we can consider the sequence of hypotheses that the algorithm produces, writing $R_n = \mathcal{A}(w_1, \dots, w_n)$ for the $n$th such hypothesis. The algorithm $\mathcal{A}$ is said to *identify the class* $\mathbb{L}$ *of languages in the limit from positive data* if for every $L \in \mathbb{L}$, for every presentation of $L$, there is an integer $n_0$ such that for all $n > n_0$, $R_n = R_{n_0}$ and $L = \mathcal{L}(R_{n_0})$. For $\mathbb{R}' \subseteq \mathbb{R}$ satisfying that $\mathbb{L} = \{\, \mathcal{L}(R) \mid R \in \mathbb{R}' \,\}$, one also says $\mathcal{A}$ *identifies* $\mathbb{R}'$ *in the limit from positive data*. For convenience, we often allow the learner to refer to the previous hypothesis $R_n$ for computing $R_{n+1}$ in addition to $w_1, \dots, w_{n+1}$. Obviously this relaxation does not effect the learnability of language classes. Moreover, learning algorithms in this paper compute hypotheses from a set of positive examples by identifying a sequence with the set consisting of the elements of the sequence.

We further require that the algorithm needs only polynomially bounded amounts of data and computation. De la Higuera's proposal is to measure the efficiency by a set of examples on which the learner converges to a representation of the target language [11].

**Definition 1 (de la Higuera [11]).** A representation class $\mathbb{R}$ is *identifiable in the limit from positive data with polynomial time and data* if and only if there exist two polynomials $p$ and $q$ and an algorithm $\mathcal{A}$ such that

1. Given a set $S$ of positive examples of size $\|S\| = m$, $\mathcal{A}$ returns a hypothesis in time $p(m)$,
2. For each representation $R \in \mathbb{R}$ of size $n$, there exists a *characteristic set* $CS$ of size less than $q(n)$ such that if $CS \subseteq S$, $\mathcal{A}$ returns a representation $R_0$ such that $\mathcal{L}(R) = \mathcal{L}(R_0)$.

The first condition (*polynomial updating time*) is widely accepted as a necessary condition for efficient learning. On the other hand, the second condition is somehow unsuitable as a model for efficient learning of CFGs, as this definition was initially designed for learning of regular languages. Even in a very restricted kind of CFGs[2], like very simple grammars [25], the length of a shortest string in the language cannot be bounded by any polynomial in the size of a grammar. At present there is no consensus on the most appropriate modification of this criterion for learning of CFGs. Several ideas have been formulated to tackle this problem. Carme et al. [4] count the cardinality $|CS|$ of a characteristic set instead of the size $\|CS\|$. Wakatsuki and Tomita [24] have proposed to measure the complexity of an algorithm dealing with CFGs by another parameter $\tau_G$, called *thickness*, defined by

$$\tau_G = \max\{\, |\omega(A)| \mid A \in V \,\} \text{ where } \omega(A) = \min\{\, w \in \Sigma^* \mid A \overset{*}{\underset{G}{\Rightarrow}} w \,\}$$

where "min" is with respect to $\prec$. We will show that our learning algorithm for $k, l$-substitutable context-free languages admits a characteristic set whose cardinality is bounded by a polynomial in the size of the target grammar and whose size is bounded by a polynomial in the thickness and the size of the target grammar.

We would like to remark that the notion of characteristic sets by de la Higuera [11] differs from that of *characteristic samples* by Angluin [1]. Let $K$ be a finite subset of a language $L$ and $\mathbb{L}$ a class of languages. We say that $K$ is a *characteristic sample* of $L$ with respect to $\mathbb{L}$ if it holds that

$$K \subseteq L' \text{ iff } L \subseteq L'$$

for any $L' \in \mathbb{L}$. The definition of a characteristic sample does not depend on any specific learning algorithm.

## 3 $k, l$-Substitutable Languages

**Definition 2 ($k, l$-substitutability).** Let $k$ and $l$ be nonnegative integers. A language $L$ is said to be *$k, l$-substitutable* if and only if for any $x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*$, $v \in \Sigma^k$, $u \in \Sigma^l$ such that $vy_1u, vy_2u \neq \lambda$,

$$x_1vy_1uz_1, x_1vy_2uz_1, x_2vy_1uz_2 \in L \text{ implies } x_2vy_2uz_2 \in L.$$

The notion of substitutability by Clark and Eyraud [7] is exactly $0, 0$-substitutability in this paper and the condition $vy_1u, vy_2u \neq \lambda$ is essential only when $k = l = 0$ (otherwise trivially $vy_1u, vy_2u \neq \lambda$ holds). $k, l$-substitutability says nothing about strings of length shorter than $k + l$. $L \cup \{w\}$ is $k, l$-substitutable if and only if $L - \{w\}$ is for any $L \subseteq \Sigma^*$, $w \in \Sigma^{<k+l}$ and integers $k$ and $l$.

It is obvious that if a language is $k, l$-substitutable, then it is $m, n$-substitutable for any $m \geq k$ and $n \geq l$. It is easy to see that the hierarchy is strict. For each $k, l \in \mathbb{N}$, there is a $k, l$-substitutable regular language that is $m, n$-substitutable if and only if $m \geq k$ and $n \geq l$. If a language is $k, 0$-substitutable or $0, k$-substitutable, then it is $k$-reversible. Recall that a language $L$ is $k$-reversible if and only if for any $x_1, y_1, x_2, y_2 \in \Sigma^*$ and $v \in \Sigma^k$, $x_1vy_1, x_1vy_2, x_2vy_1 \in L$ implies $x_2vy_2 \in L$ [1].

[2] One exception is subclasses of linear grammars.

**Proposition 1.** *$k,l$-substitutable languages are not closed under intersection with regular sets, union, concatenation, complement, Kleene closure ($+,*$), $\lambda$-free homomorphism, inverse homomorphism. $k,l$-substitutable languages are closed under reversal if and only if $k = l$. $k,l$-substitutable languages are closed under intersection and $\lambda$-free inverse homomorphism.*

*Proof.* Let us say that a pair of strings (called a *context*) $\langle x, z\rangle$ is *applicable* to $y$ in $L$ if and only if $xyz \in L$.

Intersection with regular sets: Let $L_0 = ae^*ce^*a \cup ae^*de^*a \cup be^*ce^*b$ and $L_1 = L_0 \cup be^*de^*b$. $L_0$ is regular and $L_1$ is $0,0$-substitutable. Clearly $L_1 \cap L_0 = L_0$ is not $k,l$-substitutable for any $k,l$. The context $\langle a, a\rangle$ is applicable to $e^kce^l$ and $e^kde^l$ in $L_0$, but $\langle b, b\rangle$ is applicable only to $e^kce^l$.

Union: Let $L_2 = ae^*ce^*a \cup ae^*de^*a$ and $L_3 = be^*ce^*b$. $L_2$ and $L_3$ are both $0,0$-substitutable, but the union $L_2 \cup L_3 = L_0$ is not $k,l$-substitutable for any $k,l$.

Concatenation: Languages $L_4 = ae^*c \cup ae^*d \cup b$ and $L_5 = e^*a \cup e^*ce^*b$ are $0,0$-substitutable. The concatenation $L_4L_5$ is not $k,l$-substitutable for any $k,l$, because $\langle a, a\rangle$ is applicable to $e^kce^l$ and $e^kde^l$, but $\langle b, b\rangle$ is applicable only to $e^kce^l$.

Complement: $L_6 = a^*b$ is $0,0$-substitutable, but the complement $\overline{L_6}$ is not $k,l$-substitutable for any $k,l$, because while $\langle b, \lambda\rangle$ is applicable to both $a^kaa^l$ and $a^kba^l$ in $\overline{L_6}$, $\langle \lambda, b\rangle$ is applicable only to $a^kba^l$.

Kleene closure: $L_7 = \{\, a^nba^n \mid n \geq 0 \,\}$ is $0,0$-substitutable, but neither $L_7^+$ nor $L_7^*$ is $k,l$-substitutable. Let $m = \max\{k, l\}$. The context $\langle a^m, a^{m+1}\rangle$ is applicable to $ba^{2m+1}b$ and $ba^mba^{m+1}b$ in $L_7^+$, but $\langle a^{m+1}, a^m\rangle$ is applicable only to $ba^{2m+1}b$ in $L_7^*$.

$\lambda$-free homomorphism: $L_8 = ae^*ce^*a \cup be^*ce^*b \cup fe^*de^*f$ is $0,0$-substi-tutable. Let $h$ be the homomorphism that is almost the identify but $h(f) = a$, i.e., $h(a) = a$, $h(b) = b$, $h(c) = c$, $h(d) = d$, $h(e) = e$, $h(f) = a$. $h(L_8) = L_0$ is not $k,l$-substitutable.

Inverse homomorphism: $L_6 = a^*b$ is $0,0$-substitutable. Let $h$ be such that $h(a) = a$, $h(b) = b$, $h(e) = \lambda$. $h^{-1}(L_6)$ is not $k,l$-substitutable, because $\langle \lambda, b\rangle$ is applicable to both $e^kee^l$ and $e^kae^l$ in $h^{-1}(L_6)$, but $\langle b, \lambda\rangle$ is applicable only to $e^kee^l$.

Reversal: If $L$ is $k,l$-substitutable, its reversal $L^R$ is trivially $l,k$-substi-tutable. It is enough to show that $k,l$-substitutable languages are not closed under reversal for $k > l \geq 0$. $L_9 = e^{k-1}ce^* \cup e^{k-1}de^* \cup ae^{k-1}ce^*a$ is $k,0$-substitutable, but its reversal $L_9^R$ is not $l,m$-substitutable for any $m < k$ and $l$. In $L_9^R$, $\langle \lambda, \lambda\rangle$ is applicable to both $e^lce^{k-1}$ and $e^lde^{k-1}$, but $\langle a, a\rangle$ is applicable only to $e^lce^{k-1}$.

Intersection: Let $L$ and $L'$ be $k,l$-substitutable. If $x_1vy_1uz_1, x_1vy_2uz_1, x_2vy_1uz_2 \in L \cap L'$ for some $v \in \Sigma^k$, $u \in \Sigma^l$ and $vy_1u, vy_2u \in \Sigma^+$, then those are in both $L$ and $L'$. Since $L$ and $L'$ are $k,l$-substitutable, $x_2vy_2uz_2$ is in both $L$ and $L'$ and thus in $L \cap L'$.

$\lambda$-free inverse homomorphism: Let $L$ be a $k, l$-substitutable language and $h$ a $\lambda$-free homomorphism. We denote $h(w)$ by $\overline{w}$ for readability. If $x_1vy_1uz_1$, $x_1vy_2uz_1, x_2vy_1uz_2 \in h^{-1}(L)$ for some $v \in \Sigma^k$, $u \in \Sigma^l$ and $vy_1u, vy_2u \in \Sigma^+$, then $\overline{x_1}\bar{v}\overline{y_1}\bar{u}\overline{z_1}, \overline{x_1}\bar{v}\overline{y_2}\bar{u}\overline{z_1}, \overline{x_2}\bar{v}\overline{y_1}\bar{u}\overline{z_2} \in L$. Since $L$ is $k, l$-substitutable and $|\bar{v}| \geq |v| = k$, $|\bar{u}| \geq |u| = l$, $|\bar{v}\overline{y_1}\bar{u}|, |\bar{v}\overline{y_2}\bar{u}| \geq 1$, we have $\overline{x_2}\bar{v}\overline{y_2}\bar{u}\overline{z_2} \in L$. This entails that $x_2vy_2uz_2 \in h^{-1}(L)$. $\square$

We are particularly concerned with $k, l$*-substitutable context-free languages* ($k, l$-SCFLs) in this paper. As Clark and Eyraud [7] conjecture that all $0, 0$-SCFLs are NTS languages (see [22, 3] for the definition and properties of NTS languages), we conjecture all $k, l$-SCFLs are NTS too. The simple NTS example $\{ a^nb^n \mid n \geq 1 \}$ presented by Clark and Eyraud as a non-$0, 0$-substitutable language is $1, 1$-substitutable. The class of very simple languages is also an important subclass of CFLs due to the efficient identifiability in the limit from positive data [25, 26]. Clark and Eyraud show that the class of very simple languages and that of $0, 0$-SCFLs are incomparable. It is also the case for $k, l$-SCFLs. The language generated by the very simple grammar $G$ consisting of two rules $S \to aSS$ and $S \to b$ is not $k, l$-substitutable for any $k, l$.

We note that Proposition 1 holds of classes of $k, l$-SCFLs except that $k, l$-SCFLs are not closed under intersection.

## 4 Learning Algorithm for $k, l$-Substitutable Context-Free Languages

Let us arbitrarily fix nonnegative integers $k$ and $l$. Our learning target is the class of all $k, l$*-substitutable context-free languages* ($k, l$-SCFLs). However we do not yet have any grammatical characterization of this class. For mathematical completeness, yet we have to define our learning target by saying that our target representations are CFGs generating $k, l$-substitutable languages, though this property is not decidable. We remark that the class $\{ L \mid L$ is a $k, l$-SCFL for some $k, l \in \mathbb{N} \}$ is not identifiable in the limit from positive data, because this class is superfinite modulo $\lambda$, that is, it contains at least one infinite language and all the finite languages that do not contain $\lambda$. Obviously the absence of $\lambda$ does not effect Gold's theorem [9] that any superfinite class is not identifiable in the limit from positive data.

Our learning algorithm for $k, l$-SCFLs is a natural generalization of Clark and Eyraud's original algorithm for $0, 0$-SCFLs [7]. However we omit the procedure in the original algorithm that constructs "the substitution graph" where potential nonterminal symbols that generate the same languages are merged. Though the procedure is important for making the output grammar more compact, we present a simpler learning algorithm and a simpler proof for the learnability instead.

Algorithm 1 is our learning algorithm $k, l$-SGL ($k, l$-Substitutable Grammar Learner) for learning $k, l$-SCFLs. If the new positive example is generated by the previous hypothesis by $k, l$-SGL, it keeps the hypothesis. Otherwise, let $K$ be the set of positive examples given so far. $k, l$-SGL computes the following CFG $\hat{G} = \langle \Sigma, V_K, P_K, S \rangle$ defined by

$$\begin{aligned} V_K &= \{\, [y] \mid xyz \in K,\ y \neq \lambda \,\} \cup \{S\}, \\ P_K &= \{\, [vyu] \to [vy'u] \mid xvyuz, xvy'uz \in K,\ |v| = k,\ |u| = l,\ vyu, vy'u \neq \lambda \,\} \\ &\quad \cup \{\, S \to [w] \mid w \in K \,\} \\ &\quad \cup \{\, [xy] \to [x][y] \mid [xy], [x], [y] \in V_K \,\} \\ &\quad \cup \{\, [a] \to a \mid a \in \Sigma \,\}. \end{aligned}$$

We note that $k, l$-SGL is specific to fixed nonnegative integers $k$ and $l$. In other words, $k$ and $l$ are known to $k, l$-SGL a priori.

**Algorithm 1.** $k, l$-SGL

```
Data: A sequence of strings $w_1, w_2, \dots$
Result: A sequence of CFGs $G_1, G_2, \dots$
let $\hat{G}$ = CFG generating the empty language;
for $n = 1, 2, \dots$ do
  read the next string $w_n$;
  if $w_n \notin \mathcal{L}(G)$ then
    let $\hat{G} = \langle \Sigma, V_K, P_K, S \rangle$ where $K = \{w_1, \dots, w_n\}$;
  end if
  output $\hat{G}$;
end for
```

This section will establish the following main theorem of this paper.

**Theorem 1.** *The learning algorithm $k, l$-SGL identifies $k, l$-SCFLs in the limit from positive data with polynomial updating time. $k, l$-SGL admits a characteristic set $K_G$ of polynomial cardinality in $\|G\|$ and of polynomial size in $\|G\|\tau_G$ for the target grammar $G$.*

### 4.1 Proof That Hypothesized Language Is Not Too Large

First of all we shall show that $k, l$-SGL never hypothesizes too large a language.

**Lemma 1.** *If $K$ is a finite subset of a $k, l$-substitutable language $L$, then $\mathcal{L}(\hat{G}) \subseteq L$.*

*Proof.* Let $\overline{(\cdot)}$ be the homomorphism from $(\Sigma \cup V_K - \{S\})^*$ to $\Sigma^*$ such that $\bar{a} = a$ for all $a \in \Sigma$ and $\overline{[w]} = w$ for all $[w] \in V_K - \{S\}$. We prove by induction on the length of derivation that $S \Rightarrow^+_{\hat{G}} \alpha \in (\Sigma \cup V_K - \{S\})^*$ implies $\bar{\alpha} \in L$. Suppose that the last rule used in the derivation is of the form $S \to [w]$. Then $\overline{[w]} = w \in K \subseteq L$ by definition. Suppose that $S \Rightarrow^+_{\hat{G}} \alpha B \gamma \Rightarrow \alpha\beta\gamma$ for some rule $B \to \beta$ with $B \neq S$. The only nontrivial case is when $B = [vyu]$ and $\beta = [vy'u]$ for some $v \in \Sigma^k$, $u \in \Sigma^l$ and $y, y' \in \Sigma^*$. In this case, there are $x, z \in \Sigma^*$ such that $xvyuz, xvy'uz \in K \subseteq L$ by the definition of $\hat{G}$. By induction hypothesis, we have $\overline{\alpha B \gamma} = \bar{\alpha} vyu \bar{\gamma} \in L$. Since $L$ is $k, l$-substitutable, this entails that $\bar{\alpha} vy'u \bar{\gamma} = \overline{\alpha\beta\gamma} \in L$. □

For some finite language $K$, $\hat{G}$ does not define a $k, l$-substitutable language.

*Example 1.* Let $k = l = 0$ and $K = \{ a, ab, abbc \}$. Because $a$ and $ab$ occur in the same context $\langle \lambda, \lambda \rangle$, if $L$ is a $0, 0$-substitutable language including $K$, then $L$ is closed under substituting $a$ for $ab$ and we have $abc, ac \in L$ by $abbc \in L$. On the other hand, the output grammar $\hat{G}$ by the algorithm for the input $K$ is equivalent to the grammar $G$ consisting of the following rules:

$$S \to A,\ A \to a \mid AB \mid ABc,\ B \to b \mid BBc.$$

We have $ac \in L - \mathcal{L}(G)$. That is, $\mathcal{L}(\hat{G})$ is not $0, 0$-substitutable.

Actually the least $0, 0$-substitutable language including $K$ of the above example is $a\{b, c\}^*$, which is indeed context-free and thus in the target class of our algorithm $0, 0$-SGL. This means that even if a characteristic sample (in Angluin's sense [1]) of the target $0, 0$-SCFL is given, our and Clark and Eyraud's learning algorithms do not necessarily converge to the target language.

## 4.2 Proof That Hypothesized Language Is Large Enough

To prove that the hypothesized language is large enough, we first need to define a characteristic set, that is to say a subset of a target language $L_*$ which will ensure that the algorithm $k, l$-SGL will output a grammar $\hat{G}$ such that $\mathcal{L}(\hat{G}) = L_*$. We define a characteristic set in terms of a CFG in the following normal form, while we do not yet have any grammatical characterization on CFGs generating $k, l$-substitutable languages. Because Clark and Eyraud have already given a characteristic set of $0, 0$-SCFLs for their algorithm and it works for our algorithm $0, 0$-SGL, which is essentially the same as theirs, hereafter (including the next subsection) we target $k, l$-SCFLs with $\langle k, l \rangle \neq \langle 0, 0 \rangle$.

**Definition 3.** Let $k$ and $l$ be nonnegative integers such that at least one of them is not zero. We say that a CFG $G = \langle \Sigma, V, P, S \rangle$ is in $k, l$-GNF if every production has the form $A \to w$ for some $w \in \Sigma^{\leq k+l} - \{\lambda\}$ or $A \to x\alpha z$ for some $x \in \Sigma^k$, $z \in \Sigma^l$ and $\alpha \in V^+$.

The notion of $k, l$-GNF is a generalization of Greibach normal form [10] and double Greibach normal form [20, 8]. Standard Greibach normal form is $1, 0$-GNF and double Greibach normal form is $1, 1$-GNF.

**Lemma 2.** *Let $k$ and $l$ be nonnegative integers such that at least one of them is not zero. For any* CFG *$G$, there is an equivalent* CFG *$G' = \langle \Sigma, V', P', S' \rangle$ in $k, l$-*GNF *such that $P' \subseteq V' \times (\Sigma^{\leq k+l} \cup \Sigma^k V'^{\leq 7(k+l)} \Sigma^l)$, $\|G'\|$ is polynomial in $\|G\|$ and $\tau_{G'}$ is polynomial in $\|G\|\tau_G$.*

*Proof.* CASE 1. $k, l \neq 0$. We would like to refer the reader to Engelfriet's conversion to double Greibach normal form of CFGs [8]. Observing his proof, one can see that every CFG in Chomsky normal form can be converted into an equivalent CFG whose productions have one of the following forms:

$$A \to a \text{ or } A \to a\alpha b$$

for some $A \in V$, $a, b \in \Sigma$ and $\alpha \in V^{\leq 7}$. Moreover, the size of the obtained grammar by his conversion is bounded by a polynomial in the size of the original grammar. Together with the well-known fact that any CFG can be transformed into Chomsky normal form of polynomial size, we may assume that $G = \langle \Sigma, V, P, S\rangle$ satisfies $P \subseteq V \times (\Sigma \cup \Sigma V^{\leq 7}\Sigma)$ without loss of generality.

Here we introduce a subrelation $\Rrightarrow$ of $\Rightarrow_G$. We write $\alpha \Rrightarrow \beta$ if either

- $\alpha = xA\delta$ and $\beta = x\gamma\delta$ for some $x \in \Sigma^{<k}$, $A \to \gamma \in P$ and $\delta \in (\Sigma \cup V)^*$,
- $\alpha = \delta Ax$ and $\beta = \delta\gamma x$ for some $x \in \Sigma^{<l}$, $A \to \gamma \in P$ and $\delta \in (\Sigma \cup V)^*$.

Let us define a CFG $G' = \langle \Sigma, V, P', S\rangle$ with

$$P' = \{\, A \to \alpha \mid A \overset{+}{\Rrightarrow} \alpha \in \Sigma^+ \cup (\Sigma^k(\Sigma \cup V)^*\Sigma^l) \,\}$$

where $\overset{+}{\Rrightarrow}$ is the transitive closure of $\Rrightarrow$. Some productions in $P'$ may violate the condition of $k, l$-GNF, as some terminal symbols occur in $\alpha$ in a rule of the form $A \to x\alpha z$ with $x \in \Sigma^k$ and $z \in \Sigma^l$. A solution is trivial. For each terminal symbol $a \in \Sigma$, let us introduce a new nonterminal symbol $N_a$ and a new production $N_a \to a$. Then we replace violating occurrences of terminal symbols $a$ in productions by $N_a$. It is easy to see that $\mathcal{L}(G') = \mathcal{L}(G)$.

We evaluate the size of $G'$. Because $G$ is in 1, 1-GNF, if $\alpha \Rrightarrow \beta$ and $\alpha \in \Sigma^m(\Sigma \cup V)^*\Sigma^n$, then either $\beta \in \Sigma^{m+1}(\Sigma \cup V)^*\Sigma^n$ or $\beta \in \Sigma^m(\Sigma \cup V)^*\Sigma^{n+1}$. Therefore, when $A$ has $n$ derivation steps induced by $\Rrightarrow$, i.e., $A \Rrightarrow \alpha_1 \Rrightarrow \ldots \Rrightarrow \alpha_n$, we have $n < k + l$ (note $\alpha_1 \in \Sigma V^+\Sigma \cup \Sigma^+$). Because the maximum length of production rules in $P$ is at most 9, $\alpha \Rrightarrow \beta$ implies $|\beta| \leq |\alpha| + 8$. Thus if $A \Rrightarrow^+ \alpha$, then $|\alpha| \leq 1 + 8(k + l - 1) = 8(k + l) - 7$. If $\alpha = v\alpha' u$ for some $v \in \Sigma^k$ and $u \in \Sigma^l$, then $|\alpha'| \leq 7(k + l - 1)$. Moreover we see that $|P'| \leq |P|^{k+l-1} + |\Sigma|$. We have $\|G'\| \leq (8(k + l) - 7)(|P|^{k+l-1} + |\Sigma|) \in O(|P|^{k+l-1})$.

Moreover, it is not hard to see that when Engelfriet's conversion is applied to a CFG in Chomsky normal form obtained from a general CFG $G''$ by a reasonable method, then the thickness $\tau_G$ of the resultant grammar $G$ in 1, 1-GNF is bounded by a polynomial in $\|G''\|\tau_{G''}$. By the fact $\tau_{G'} = \tau_G$, we get the lemma.

CASE 2. $k > 0$ and $l = 0$. Apply the similar conversion to Case 1 to CFG $G$ in Greibach normal form such that $P \subseteq V \times \Sigma V^{\leq 2}$.

CASE 3. $k = 0$ and $l > 0$. This case is just symmetric to Case 2. □

It is easy to get rid of useless nonterminals in $G'$ obtained by the above method if any.

Now we define a characteristic set $K_G$ of a $k, l$-SCFL in terms of a reduced CFG $G = \langle \Sigma, V, P, S\rangle$ in $k, l$-GNF generating it as follows, where "min" is with respect to $\prec$, which is extended from $\Sigma^*$ to $\Sigma^* \times \Sigma^*$ in some reasonable way:

$$\omega(\alpha) = \min\{\, w \in \Sigma^* \mid \alpha \overset{*}{\underset{G}{\Rightarrow}} w \,\} \text{ for } \alpha \in (\Sigma \cup V)^*,$$

$$\chi(A) = \min\{\, \langle x, z\rangle \in \Sigma^* \times \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} xAz \,\} \text{ for } A \in V,$$

$$\begin{aligned} K_A &= \{\, vw_1 \dots w_n u \in \Sigma^* \mid A \to vB_1 \dots B_n u, B_i \to \beta_i \in P,\, w_i = \omega(\beta_i) \,\} \\ &\quad \cup \{\, y \in \Sigma^* \mid A \to y \in P \,\} \text{ for } A \in V, \\ K_G &= \{\, xyz \in \Sigma^* \mid \chi(A) = \langle x, z \rangle,\, y \in K_A,\, A \in V \,\}. \end{aligned}$$

The following trivial lemma is implicitly used in the proof of Lemma 4.

**Lemma 3.** *$K_A \subseteq \mathcal{L}(G, A)$ and $K_S \subseteq K_G \subseteq \mathcal{L}(G)$. $K_G$ is finite.*

*Let $k, l$-SGL compute $\hat{G} = \langle \Sigma, V_K, P_K, S \rangle$ from $K$ such that $K_G \subseteq K \subseteq \mathcal{L}(G)$. Then for any $w \in K_A$, $[w] \in V_K$. If $[w_1 \dots w_m] \in V_K$ with $w_1, \dots, w_m \in \Sigma^+$, then $[w_1 \dots w_m] \Rightarrow^*_{\hat{G}} [w_1] \dots [w_m] \overset{*}{\Rightarrow} w_1 \dots w_m$.*

**Lemma 4.** *Suppose that the algorithm outputs $\hat{G}$ for the input $K$ including $K_G$. Then $\mathcal{L}(G) \subseteq \mathcal{L}(\hat{G})$.*

*Proof.* We first show that if $A \to vB_1 \dots B_n u \in P$ with $v \in \Sigma^k$, $u \in \Sigma^l$ and $w_i \in K_{B_i}$, then there is $w \in K_A$ such that $[w] \Rightarrow^*_{\hat{G}} v[w_1] \dots [w_n]u$. Let $\beta_i$ be such that $B_i \Rightarrow_G \beta_i \overset{*}{\Rightarrow} w_i$ and

$$I = \{\, i \mid w_i \neq \omega(\beta_i),\, 1 \leq i \leq n \,\}.$$

For each $i \in I$, we have $\beta_i \in \Sigma^k V^+ \Sigma^l$. Thus there are $v_i \in \Sigma^k$, $u_i \in \Sigma^l$ and $y_i, y'_i \in \Sigma^*$ such that $w_i = v_i y_i u_i$ and $\omega(\beta_i) = v_i y'_i u_i$. The fact $\omega(\beta_i), w_i \in K_{B_i}$ entails that $x_i v_i y'_i u_i z_i, x_i v_i y_i u_i z_i \in K_G$ where $\langle x_i, z_i \rangle = \chi(B_i)$. By definition, $\hat{G}$ has rule $[\omega(\beta_i)] \to [w_i] \in P_K$. We have $v\omega(\beta_1) \dots \omega(\beta_n)u \in K_A$ and

$$[v\omega(\beta_1) \dots \omega(\beta_n)u] \underset{\hat{G}}{\overset{*}{\Rightarrow}} v[\omega(\beta_1)] \dots [\omega(\beta_n)]u \overset{*}{\Rightarrow} v[w_1] \dots [w_n]u.$$

By using this claim inductively, we see that for any $A \Rightarrow^*_G w \in \Sigma^*$, there is $w' \in K_A$ such that $[w'] \Rightarrow^*_{\hat{G}} w$. Since $\hat{G}$ has rule $S \to [w'] \in P_K$ for any $w' \in K_S$, we obtain the lemma. □

Clark and Eyraud [7] define a characteristic set of $0, 0$-SCFLs $\mathcal{L}(G)$ by

$$CS(G) = \{\, xyz \mid A \to \beta \in P,\, \langle x, z \rangle \in \chi(A),\, y = \omega(\beta) \,\},$$

where $G$ is not assumed to be in any special form. This set $CS(G)$ is more compact than $K_G$. However, $CS(G)$ can be too small as a characteristic set of a $k, l$-SCFL in general. Let $G$ be a CFG in $1, 0$-GNF consisting of production rules $S \to aSC$, $S \to b$ and $C \to c$. Then $\mathcal{L}(G) = \{\, a^n b c^n \mid n \geq 0 \,\}$ is $1, 0$-substitutable. On the other hand, $CS(G) = \{\, b, abc \,\}$ is also $1, 0$-substitutable, and thus $CS(G)$ cannot be a characteristic set of $\mathcal{L}(G)$ for any algorithm learning $1, 0$-SCFLs.

### 4.3 Polynomial Time and Data

Now we discuss the efficiency of our learning algorithm $k, l$-SGL. Though the class of $k, l$-SCFLs is not identifiable in the limit from positive data with polynomial time and data in de la Higuera's sense (Definition 1), $k, l$-SGL satisfies de la Higuera's definition if we accept the thickness $\tau_G$ of the target grammar as a fundamental parameter (Lemma 7). Besides, $k,l$-SGL identifies $k,l$-SCFLs in the limit from positive data with polynomial time and data in Carme et al.'s sense [4], i.e., $k,l$-SGL admits a characteristic set of polynomial cardinality (Lemma 6). Although we have no grammatical characterization of $k,l$-SCFLs, Lemma 2 justifies evaluating the characteristic set $K_G$ where $G$ is in $k,l$-GNF such that $P \subseteq V \times (\Sigma^{\leq k+l} \cup \Sigma^k V^{\leq 7(k+l)} \Sigma^l)$.

**Lemma 5.** *Computation of $\hat{G}$ from a finite language $K$ is done in polynomial time in the description size of $K$.*

*Proof.* Let $\ell_K = \max\{\, |w| \mid w \in K \,\}$. For fixed $w, w' \in K$, the cost for enumerating all pairs $vyu$ and $vy'u$ such that $w = xvyuz$, $w' = xvy'uz$, $|v| = k$, $|u| = l$, $vyu, vy'u \neq \lambda$ for some $x, z \in \Sigma^*$ is bounded by $O(\ell_K^2)$. Thus computing all the rules of the form $[vyu] \to [vy'u]$ takes $O(|K|^2 \ell_K^2)$ time. Computing all the rules of the form $S \to [w]$ for $w \in K$ takes $O(\|K\|)$ time. For each $[w] \in V_K$, there are $(|w|-1)$ pairs $\langle x, y \rangle$ such that $w = xy$ and $x, y \neq \lambda$. Thus computing all the rules of the form $[xy] \to [x][y]$ takes $O(|V_K|\ell_K)$ time. Together with $\ell_K, |K| \leq \|K\|$ and $|V_K| \leq \|K\|^2$, totally the algorithm updates its hypothesis in $O(\|K\|^4)$ time. $\square$

Therefore, $k,l$-SGL updates its hypothesis quickly even for large $k$ and $l$. However, the amount of data for letting $k,l$-SGL converge increases depending on $k$ and $l$. For instance, to learn the $k,l$-SCFL $\Sigma^{\leq k+l} - \{\lambda\}$, the learner requires all elements of $\Sigma^{\leq k+l} - \{\lambda\}$ to be given as positive examples, because any subset of $\Sigma^{\leq k+l} - \{\lambda\}$ is also a $k,l$-SCFL.

**Lemma 6.** *$|K_G|$ is bounded by a polynomial in $\|G\|$.*

*Proof.* Let $n = \max\{\, |\beta| \mid A \to x\beta z \in P \text{ with } |x| = k,\ |z| = l \,\}$. Then we have $|K_G| \leq |P|^{n+1}$. By Lemma 2, we have $n \leq 7(k+l)$ (constant). $|K_G|$ is bounded by a polynomial. $\square$

**Lemma 7.** *The description size of $\|K_G\|$ is bounded by a polynomial in $\|G\|$ and $\tau_G$.*

*Proof.* By Lemma 6, it is enough to prove that the length of each element in $K_G$ is bounded by a polynomial in $\|G\|$ and $\tau_G$. Suppose that $xyz \in K_G$ where $\chi(A) = \langle x, z \rangle$ and $y \in K_A$ for $A \in V$. We have a derivation

$$A_0 \underset{G}{\Rightarrow} \alpha_1 A_1 \gamma_1 \Rightarrow \cdots \Rightarrow \alpha_1 \ldots \alpha_m A_m \gamma_m \ldots \gamma_1 \overset{*}{\Rightarrow} x A_m z$$

where $A_0 = S$, $A_{i-1} \to \alpha_i A_i \gamma_i$ for $i = 1, \ldots, m$, $A_m = A$, $x = \omega(\alpha_1 \ldots \alpha_m)$ and $z = \omega(\gamma_m \ldots \gamma_1)$. We see $A_i \neq A_j$ if $i \neq j$ by the definition of $\chi(A)$. Thus $|\alpha_1 \ldots \alpha_m \gamma_m \ldots \gamma_1| \leq \|G\|$ and $|xz| \leq \|G\|\tau_G$. If $y \in K_A$, then either $A \to y \in P$, or there are productions $A \to vB_1 \ldots B_n u \in P$, $B_i \to \beta_i \in P$ for $i = 1, \ldots, n$ and $y = v\omega(\beta_1 \ldots \beta_n)u$. Let $p = k + l$ (constant). By Lemma 2, we have $n \leq 7p$ and $\beta_i \in \Sigma^{\leq p} \cup \Sigma^k V^{\leq 7p} \Sigma^l$. Therefore $|\omega(\beta_i)| \leq p + 7p\tau_G$ and $|y| \leq p + 7p(p + 7p\tau_G) \in O(\tau_G)$. All in all we have $|xyz| \in O(\|G\|\tau_G)$. $\square$

## 5 Discussion

Following the proposal given by Clark and Eyraud [7], this paper gave a formal definition of a hierarchy of substitutable languages by generalizing the original notion of substitutability and showed that each class of context-free languages in the hierarchy is polynomial-time identifiable in the limit from positive data. While this generalization can be thought of as the exact analogue of $k$-reversibility introduced by Angluin [1], some properties that hold of $k$-reversible regular languages do not hold of $k, l$-SCFLs, or are not known to hold of $k, l$-SCFLs.

One is a grammatical characterization of $k, l$-SCFLs, as already pointed out by Clark and Eyraud. The original definition of $k$-reversible languages is given in terms of finite state automata and the syntactic characterization of them is a theorem [1].

Kobayashi and Yokomori [14] have shown that the least $k$-reversible language including a finite language is always regular. In fact Angluin's learning algorithm always hypothesizes the least $k$-reversible regular language including the given data. On the other hand, the least $0, 0$-substitutable language including $\{abc, acb, bac, bca, aabbcc\}$ is MIX $= \{\, w \in \{a, b, c\}^+ \mid |w|_a = |w|_b = |w|_c \,\}$, which is known to be non-context-free.[3] Moreover, MIX does not have a least $0, 0$-SCFL including it. $L_1 = \{\, w \in \{a, b, c\}^+ \mid |w|_a = |w|_b \,\}$ and $L_2 = \{\, w \in \{a, b, c\}^+ \mid |w|_a = |w|_c \,\}$ are $0, 0$-SCFLs and MIX $= L_1 \cap L_2$. This shows that some set of positive examples does not admit a least consistent $0, 0$-SCFL.

The literature has established many results on reversible regular languages and their variants (e.g., [2,14,16,15,13,19,21,23]). It would be interesting to investigate whether or not analogous results hold of $k, l$-SCFLs.

Clark and Eyraud's algorithm SGL for $0, 0$-SCFLs [7] bases Clark's PAC learning algorithm for unambiguous NTS languages [5]. Though some unambiguous NTS languages are not $0, 0$-substitutable, taking into account the difference of context distributions of substrings, he successes learning non-$0, 0$-SCFLs using SGL. Our learning algorithm $k, l$-SGL is more powerful than SGL for $k, l > 0$, but we still conjecture all $k, l$-SCFLs are NTS. It is doubtful whether an application of Clark's method to $k, l$-SGL could enable a PAC learning algorithm that is more efficient or more powerful.

## Acknowledgement

The author is deeply grateful to Rémi Eyraud, Alexander Clark and Thomas Zeugmann for their valuable comments and advice. He also appreciates the anonymous reviewers for their helpful comments and suggestions.

This work was supported by Grant-in-Aid for Young Scientists (B-20700124) and a grant from the Global COE Program, "Center for Next-Generation Information Technology based on Knowledge Discovery and Knowledge Federation", from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

[3] It is Eyraud and Clark who gave the author a critical clue to find this example in personal communication.

## References

1. Angluin, D.: Inference of reversible languages. Journal of the Association for Computing Machinery 29(3), 741–765 (1982)
2. Angluin, D.: Negative results for equivalence queries. Machine Learning 5, 121–150 (1990)
3. Boasson, L., Sénizergues, G.: NTS languages are deterministic and congruential. Journal of Computer and System Sciences 31(3), 332–342 (1985)
4. Carme, J., Gilleron, R., Lemay, A., Niehren, J.: Interactive learning of node selecting tree transducer. Machine Learning 66(1), 33–67 (2007)
5. Clark, A.: PAC-learning unambiguous NTS languages. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 59–71. Springer, Heidelberg (2006)
6. Clark, A., Eyraud, R.: Identification in the limit of substitutable context-free languages. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 283–296. Springer, Heidelberg (2005)
7. Clark, A., Eyraud, R.: Polynomial identification in the limit of context-free substitutable languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
8. Engelfriet, J.: An elementary proof of double Greibach normal form. Information Processing Letters 44(6), 291–293 (1992)
9. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
10. Greibach, S.A.: A new normal-form theorem for context-free phrase structure grammars. Journal of the Association for Computing Machinery 12(1), 42–52 (1965)
11. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27, 125–138 (1997)
12. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition 38(9), 332–1348 (2005)
13. Kobayashi, S.: Iterated transductions and efficient learning from positive data: A unifying view. In: Oliveira, A.L. (ed.) ICGI 2000. LNCS (LNAI), vol. 1891, pp. 157–170. Springer, Heidelberg (2000)
14. Kobayashi, S., Yokomori, T.: On approximately identifying concept classes in the limit. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS, vol. 997, pp. 298–312. Springer, Heidelberg (1995)
15. Kobayashi, S., Yokomori, T.: Identifiability of subspaces and homomorphic images of zero-reversible languages. In: Li, M., Maruoka, A. (eds.) ALT 1997. LNCS, vol. 1316, pp. 48–61. Springer, Heidelberg (1997)
16. Kobayashi, S., Yokomori, T.: Learning approximately regular languages with reversible languages. Theoretical Computer Science 174(1-2), 251–257 (1997)
17. Lange, S., Zeugmann, T., Zilles, S.: Learning indexed families of recursive languages from positive data: A survey. Theoretical Computer Science 397(1-3), 194–232 (2008)
18. Lee, L.: Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Harvard University (1996), `ftp://deas-ftp.harvard.edu/techreports/tr-12-96.ps.gz`
19. Mäkinen, E.: On inferring zero-reversible languages. Acta Cybernetica 14(3), 479–484 (2000)
20. Rosenkrantz, D.J.: Matrix equations and normal forms for context-free grammars. Journal of ACM 14(3), 501–507 (1967)

21. Sempere, J.M.: Learning reversible languages with terminal distinguishability. In: Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E. (eds.) ICGI 2006. LNCS (LNAI), vol. 4201, pp. 354–355. Springer, Heidelberg (2006)
22. Sénizergues, G.: The equivalence and inclusion problems for NTS languages. Journal of Computer and System Sciences 31(3), 303–331 (1985)
23. Tîrnauca, C., Knuutila, T.: Polynomial time algorithms for learning k-reversible languages and pattern languages with correction queries. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 272–284. Springer, Heidelberg (2007)
24. Wakatsuki, M., Tomita, E.: A fast algorithm for checking the inclusion for very simple deterministic pushdown automata. IEICE transactions on information and systems E76-D(10), 1224–1233 (1993)
25. Yokomori, T.: Polynomial-time identification of very simple grammars from positive data. Theoretical Computer Science 298, 179–206 (2003)
26. Yokomori, T.: Erratum to Polynomial-time identification of very simple grammars from positive data. Theoret. Comput. Sci. 298, 179–206 (2003); Theoretical Computer Science 377(1-3), 282–283 (2007)

# Learning Subclasses of Pure Pattern Languages

P.J. Abisha[1], D.G. Thomas[1], and Sindhu J. Kumaar[2]

[1] Department of Mathematics, Madras Christian College, Chennai - 600 059
[2] Department of Mathematics, Crescent Engineering College, Chennai - 600 048
sindhujkumaar@yahoo.co.in

**Abstract.** Pattern language learning algorithms within the inductive inference model and query learning setting have been of great interest. In this paper, we study the problem of learning pure pattern languages using queries and examples.

## 1 Introduction

Inductive inference introduced by Gold [6], is a model that treats as an infinite process, which identifies the unknown concept in the limit. Inferring a pattern common to all words in a given sample is a typical instance of inductive inference. Motivated by the study of Angluin [2], a generative device called pattern grammar is defined by Dassow et al. [5]. In [1], a new generative device called the pure pattern grammar is defined. We do not specify variables, instead constants themselves are replaced by axioms initially and the process is continued with the current set of words to get the associated language. As the study of pattern languages is motivated by the inference problem and there are algorithms to learn subclasses of pure languages, it is of interest to analyze the inference problem for pure pattern languages. Here we give algorithms to learn two subclasses of the family of pure pattern languages using queries.

## 2 Pure Pattern Grammars

**Definition 1.** *A pure pattern grammar is a triple $G = (\Sigma, A, P)$ where $\Sigma$ is an alphabet, $A \subseteq \Sigma^*$ is a finite set of elements of $\Sigma^*$ called axioms and $P$ is a finite subset of $\Sigma^+$ called the set of patterns. For a set $P$ and a language $L \subseteq \Sigma^*$, let $P(L)$ be the set of strings obtained by replacing, uniformly and in parallel each letter of the patterns in $P$, by strings in $L$, occurrences of the same letter of a pattern in a particular step being replaced by the same string. The language (PPL) generated by $G$, denoted by $L(G)$, is the smallest $L \subseteq \Sigma^*$ for which we have $P \subseteq L$, $A \subseteq L$ and $P(L) \subseteq L$. Here $L(G) = P \cup A \cup P(A) \cup P(P(A)) \cup \ldots$.*

**Proposition 1.** *(i) Any finite set $F$ with at least one word $p$ of length 1 is a pure pattern language, (ii) The families of pure context free languages (PCF), context free languages (CFL), regular languages (RL) are incomparable with the family of pure pattern languages (PPL). The family of pure pattern languages generated by grammars with a single pattern is strictly included in the family of deterministic tabled 0L languages.*

## 3 Learning a Subclass of PPL

We now give a polynomial time algorithm to learn a subclass of PPL using the restricted subset queries and restricted superset queries. The inclusion problem for this class is decidable, as the difference between the pattern languages with a single pattern over only variables and the pure pattern languages of this class lies mainly in the patterns. The technique of the algorithm is as follows: First, the pattern is learnt using restricted superset queries and then the axioms are learnt using restricted subset queries. We assume that length of the pattern is known (fixed), say '$n$' and the length of the longest axioms is known, say '$m$'. If $\Sigma = \{a_1, a_2, \ldots, a_k\}$ is the known alphabet of the language to be learnt then the axiom set $A$ is a subset of all words over $\Sigma$ of length $i$, $1 \leq i \leq m$. The words in $\Sigma^n$ are lexicographically arranged and restricted superset queries for $\left(\Sigma, \bigcup_{i=0}^{m} \Sigma^i, p_i\right)$ where $p_i \in \Sigma^n$ are made. If the answer is yes, $p_i$ is the pattern $p$; otherwise repeat the same procedure for the next word in $\Sigma^n$. Now, to learn axiom set $A$, initially fix $A = \phi$. Arrange the words in $\Sigma^i$, $i = 0$ to $m$, according to increasing order of length and among the words of equal length lexicographically. Let them be $w_1, w_2, \ldots, w_s$. At the $t^{th}$ step, ask the restricted subset query for $(\Sigma, A \cup \{w_t\}, p)$. If the answer is 'yes', increment $A$ to $A \cup \{w_t\}$. If the answer is 'no', $A$ is not incremented. The output at the last step is the required PPG.

```
Algorithm
Input:   An alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$, $m = max\{|x_i| : x_i \in A\}$, $n = |p|$
         Words $p_1, p_2, \ldots, p_r$ from $\Sigma^n$ arranged lexicographically.
         Words $w_1, w_2, \ldots, w_s$ are given in the increasing length order, among
         words of equal length according to lexicographic order.
Output:  $G = (\Sigma, A, p)$
begin
    for $t = 1$ to $r$ do
        begin
            ask restricted superset query for $\left(\Sigma, \bigcup_{i=0}^{m} \Sigma^i, p_t\right)$
            If 'yes' then output $p = p_t$
                else $t = t + 1$
        end
    $A = \phi$
    for $t = 1$ to $s$ do
    begin
        ask restricted subset query for $G = (\Sigma, A \cup \{w_t\}, p)$
        If 'yes' then $A = A \cup \{w_t\}$ and $t = t + 1$
            else output $G$
    end
end
```

## 4 Learning Another Subclass of PPL

In the MAT learning [3], the teacher / oracle can answer membership query and equivalence query. We consider any subclass of PPL for which membership and equivalence queries are decidable. In addition, we require that, the axiom set $A$ is a code [4]. However, the PPG need not have only a single pattern. The algorithm to learn a pure pattern language $L(G)$ from the above subclass works as follows:

The fixed alphabet $\Sigma$ of cardinality $k$ and the axiom set $A$ whose cardinality is greater than or equal to $k$ are the inputs to the algorithm. Let the target PPG be $G = (\Sigma, A, P)$. Initially, when $j = 0$, the pattern set $P_j$ is assumed to be empty by the algorithm. The learner asks the oracle, first the equivalence query for $L(G)$ and $L(G_j)$ where $G_j = (\Sigma, A, P_j)$. We have either $L(G_j) \subset L(G)$ or $L(G_j) = L(G)$. If the answer is positive, we obtain an equivalent grammar $G_j$ of the target grammar. Otherwise, a positive sample word $x \in L(G) - L(G_j)$ is returned. If $x \notin A^+$. Then $P_{j+1} = P_j \cup \{x\}$ and $G_{j+1} = (\Sigma, A, P_{j+1})$. If $x \in A^+$, then the learner factorises $x$ over $A$. It should be noted that this can be done in linear time by Sardinas-Patterson algorithm [4]. Let $x = x_1x_2x_3 \ldots x_m$, $x_i \in A$. The learner finds the minimal prefix of $x$ which belongs to $L(G)$. It is denoted by $min(x)$ and this is found by asking membership query to the oracle. Let $min(x) = w = x_1x_2 \ldots x_r$ and $bagmin(x) = \{x'_1x'_2 \ldots x'_s\}$ where $x'_j$ $(1 \leq j \leq s)$ are distinct and $\{x_1, x_2, \ldots, x_r\} = \{x'_1, x'_2, \ldots, x'_s\}$. It should be noted that the number of distinct elements in $bagmin(x)$ is less than or equal to number of elements in $A$. Let $\{f_1, f_2, \ldots, f_n\}$ be the set of all 1-1 morphisms from $bagmin(x)$ to $\Sigma$. There is atmost $k!$ such morphisms (i.e.,) $n \leq k!$. The learner asks membership query for $f_q(min(x))$ $(q = 1, 2, \ldots, n)$. If the answer is positive the learner updates the pattern set $P_j$ to $P_{j+1} = P_j \cup \{f_q(min(x))\}$ and asks the equivalence query for $L(G)$ and $L(G_{j+1})$, where $G_{j+1} = (\Sigma, A, P_{j+1})$. This process is repeated until we get a PPG equivalent to $G$.

## References

1. Abisha, P.J., Subramanian, K.G., Thomas, D.G.: Pure Pattern Grammars. In: Proceedings of International Workshop on Grammar Systems, Austria, pp. 253–262 (2000)
2. Angluin, D.: Finding patterns common to a set of strings. Journal of Computer and System Sciences 21, 46–62 (1980)
3. Angluin, D.: Learning regular sets from queries and counter examples. Information and Computation 75, 87–106 (1987)
4. Berstel, J., Perrin, D.: The Theory of Codes. Academic Press, New York (1985)
5. Dassow, J., Paun, G., Salomaa, A.: Grammars based on patterns. International Journal of Foundations of Computer Science 4, 1–14 (1993)
6. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)

# Which Came First, the Grammar or the Lexicon?

Tom Armstrong[1] and Tim Oates[2]

[1] Wheaton College
Norton, MA 02766 USA
armstrong_tom@wheatoncollege.edu
[2] University of Maryland Baltimore County
Baltimore, MD 21250 USA
oates@cs.umbc.edu

**Abstract.** Computational approaches to learning aspects of language typically reduce the problem to learning syntax alone, or learning a lexicon alone. These simplifications have led to disconnected solutions and some unreasonable assumptions about inputs to their algorithms. In this paper, we present an approach that exploits a grammar learning algorithm to learn its own alphabet, or lexicon. We present empirical results and categorize the successes and types of errors lexical acquisition approaches encounter.

**Keywords:** lexical learning, applications of grammars, natural language learning.

## 1 Introduction

Many grammar learning algorithms derive inspiration from the distinctly human ability to learn language. While children are facile at learning an alphabet (i.e., words in the language composed on phonemes) and constructing a generative grammar using that alphabet, our computational approaches are often brittle and prone to make unrecoverable errors. Learning grammars is an intractable problem unless, for one, concessions are made regarding the input, and having complete knowledge of the language's alphabet is a common assumption. For example, most algorithms expect an input like *the cat hates the dog*, and not an input like *thecathatesthedog*.

This paper explores the utility of including higher-level structural information (in the form of a learned grammar) in the unsupervised learning of a lexicon. We remove the assumption that the grammar learning algorithms have perfectly segmented input data. We discuss this learning task in terms of the lexical-syntactic interface [1] where two learning tasks (i.e., lexical acquisition and grammar learning) are bootstrapped together. Here we extend our approach through experimentation with additional lexical data.

## 2 Lexical-Syntactic Interface

The lexical-syntactic interface is the interplay between the learning tasks of lexical acquisition and grammar induction. A typical lexicon learning algorithm begins with a stream of categorical data or a set of strings, and its goal is to induce an inventory of lexical items. A typical grammar induction algorithm begins with a set of strings, and its goal is to learn a generative structural model like the RPNI example above. While lexical learning is done without any regard for structural information, grammar induction assumes a known lexicon and correctly segmented input strings. In the lexical-syntactic interface, we exploit the structure inherent in the sequences of words and inside of words.

GramLex is the algorithmic instantiation of the lexical-syntactic interface in the form of a bootstrap algorithm [1]. We detailed the specific algorithmic components of the interface and presented experimental results on a variety of benchmark languages. The grammar learning community has a series of benchmark languages for comparing learning algorithms: $\mathcal{L}_1$ through $\mathcal{L}_{15}$ (Canonical deterministic finite automata and data are available from `http://www.irisa.fr/symbiose/people/coste/gi_benchs.html`) [2,3].

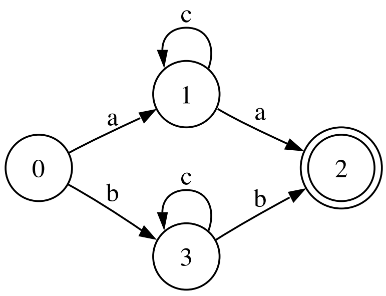


**Fig. 1.** Target Machine Topology

Let us look at an example of $\mathcal{L}_{15}$ with a random lexicon where GramLex learns a lexicon using a grammar learning algorithm. Using the three words a = *ow r ih n jh* (orange), b = *p er p ax l* (purple), and c = *r ey z ih n z* (raisins) in $\mathcal{L}_{15}$, we begin with $\Sigma$ (the alphabet) and $\Gamma$ (the initial lexicon) = $\{\lambda$, *ow*, *r*, *ih*, *n*, *jh*, *p*, *er*, *ax*, *l*, *ey*, *z*$\}$. Given a characteristic sample for this language, GramLex returns the automaton in Figure 1.

## 3 Experiments

While the grammar learning community has made an effort to evaluate algorithms empirically, it is less obvious that the lexicon learning community has done the same. Proper evaluation of our lexical and grammatical bootstrap is a challenge with respect to the lexicon we select. To begin, we choose a random collection of words found in the SWITCHBOARD corpus. Using the human-annotated phonetic transcriptions for each word, we provide the sequence of phonemes for each word (an ARPAbet symbol for each phoneme) to the bootstrap in place of the standard alphabet in the characteristic sample. We run our bootstrap algorithm and analyze the resulting learned machine. The words vary in length from a maximum length of 15 phonemes (e.g., *eh k s t r ow r d ih n eh r ax l iy* or extraordinarily) to a minimum length of 2 phonemes (e.g., *m ey* or my) and collectively had a mean length of about 7 phonemes. Here we report the results on a trial of 50 randomly selected lexicons and the language $\mathcal{L}_{15}$. Of the 50 trials, 39 learned the correct lexicon and the correct grammar.

The remaining 11 trials that did not completely learn the lexicon or the grammar are categorized into four distinct classes of errors. Class 1 errors (2/11) are trials where we correctly learn the lexicon, but not the correct grammar. Class 2 errors (2/11) are trials where we correctly learn the automaton (with one caveat), and we correctly learn the lexicon (with one caveat). That is, the grammar accept all of the strings in the language, but also accepts a special overgeneralize-type string. Class 3 errors (5/11) are trials that fail to learn the entire lexicon and overgeneralize beyond the surface equivalence found in class 2 errors. While the machine contains some correct structure, the resulting machines further and further fail the *looks good* test. Class 4 errors (2/11) are the most egregious in terms of incorrectly learned structure and incorrect lexical items.

## 4 Conclusion and Future Work

In this paper, we presented an extension to our novel framework for bootstrapping the acquisition of a lexicon and learning a grammar. Prior work on lexical acquisition has ignored how those terms are used from a syntactic point of view, and grammar learning approaches typically require perfectly formed inputs to guarantee any learning result. This work demonstrates the viability of learning both tasks in tandem for a rich diversity of languages and lexicons.

Future work will proceed in two directions. First, we will focus on futher defining the boundaries between the learnable and the pathological for certain lexicons. Second, we will expand the class of languages we are interested in learning. Context-free grammars and natural languages may provide even more structure to guide lexicon learning and, in fact, make learning easier.

## References

1. Armstrong, T., Oates, T.: Learning in the lexical-grammatical interface. In: FLAIRS Conference. AAAI Press, Menlo Park (2008)
2. Tomita, M.: Dynamic construction of finite automata from examples using hill climbing. In: Proceedings of the 4th Annual Cognitive Science Conference, pp. 105–108 (1982)
3. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the gig method. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 236–245. Springer, Heidelberg (1994)

# Learning Node Label Controlled Graph Grammars (Extended Abstract)

Christophe Costa Florêncio

Department of Computer Science, K.U. Leuven, Leuven, Belgium
Tel.: +32.(0)16327863; Fax: +32.(0)16327996
Chris.CostaFlorencio@cs.kuleuven.be

## 1 Introduction

Within the data mining community there has been a lot of interest in mining and learning from graphs (see [1] for a recent overview). Most work in this area has has focussed on finding algorithms that help solve real-world problems. Although useful and interesting results have been obtained, more fundamental issues like learnability properties have hardly been adressed yet. This kind of work also tends not to be grounded in graph grammar theory, even though some approaches aim at inducing grammars from collections of graphs.

This paper is intended as a step towards an approach that is more theoretically sound. We present results concerning learnable classes of graph grammars.

## 2 Graph Grammars

Many approaches to representing graph languages exist, the present paper is restricted to the popular node label controlled (*NLC*) grammars. These consist of production rules with a non-terminal label at the left-hand side (lhs) of a rule, and on the right-hand side (rhs) a graph called the *daughter graph*, and an *embedding relation*. The daughter graph has its nodes labelled with both terminal and non-terminal labels.

Generation of a graph starts with the *axiom*, a node labelled with the start non-terminal symbol. Rules from the grammar are applied such that non-terminal nodes are replaced by daughter graphs, which are connected to the *host graph* according to the embedding relations. The graph language generated by a grammar consists of all graphs thus obtained that have terminal labels exclusively.

The embedding relation specifies how the daughter graph is connected by considering just the *neighbourhood* of the replaced node. For each vertex in the daughter graph, the embedding relation specifies either 'empty' or a node label. All nodes in the neighbourhood with the label will be connected to that vertex.

We assume that all rules in all grammars are productive, i.e., do not contain useless symbols, and that unit- and $\epsilon$-productions are absent. We also assume that every rule contains at least one terminal (cf lexicalized TAG, for example).

Note that for many classes of graph grammar the generating grammar does not necessarily function as a parser as well. The reason is that, as part of a derivation step, the edges incident on the node to be replaced are removed, and the daughter graph that is inserted is connected in a pre-determined way. In this setting, there is no way to recover the removed edges. This may however be required for deciding membership of some given graph.

A number of restricted subclasses of *NLC* grammars can be found in the literature, we will focus on Boundary *NLC* (*B-NLC*), which disallows edges between non-terminal vertices in the rhs. This is the most expressive class of NLC grammars known for which parsers can effectively be obtained.

The *graph language generated by* grammar $G$ will be denoted $GL(G)$, the *derivation language generated by* grammar $G$ will be denoted $DL(G)$.

Informally speaking, a *derivation tree* for a graph and graph grammar is a tree that reflects the generation process from grammar to derived graph, i.e., the nodes correspond to the applications of rewrite rules. We define them so that the nodes are labelled with the daughter graphs of their corresponding rules. The daughters of any node in the tree correspond to the rewritings applied to the non-terminals in the rhs. The number of daughters is exactly the number of non-terminals, that is, these rules are considered to all be applied in one step.

In the case of a rule that has no non-terminals in the rhs (a *terminal* rule), the corresponding node is a leaf. In the present context, the embedding relations can be safely ignored, we thus leave these out of the derivation tree representation.

## 3 Learnability

We are interested in learnability in the technical sense of identification in the limit ([2]). In this paradigm a class of languages is considered learnable just if there exists an algorithm over sequences of input data that converges on a correct hypothesis after a finite number of presentations. It is assumed that all data is presented eventually. A sufficient condition for a class to be identifiable in the limit ([3]) is being r.e., consisting of just recursive languages and having the property of *infinite elasticity*:

**Definition 1.** ***Infinite elasticity***[3, 4]
*A class $\mathcal{L}$ of languages is said to have* infinite elasticity *if there exists an infinite sequence $\langle s_n \rangle_{n\in\mathbb{N}}$ of sentences and an infinite sequence $\langle L_n \rangle_{n\in\mathbb{N}}$ of languages in $\mathcal{L}$ such that for all $n \in \mathbb{N}$, $s_n \notin L_n$, and $\{s_0, \ldots, s_n\} \subseteq L_{n+1}$.*

*A class $\mathcal{L}$ of languages is said to have* finite elasticity *if it does not have infinite elasticity.*

So, one way of proving learnability of a class is demonstrating it has finite elasticity and to extend the class by exploiting a closure property. The following theorem, from [5], is useful when the relation between language element and possible derivation is finite-valued. It is generally easier to prove finite elasticity of a class of derivation languages than of a class of string languages.

Let $\Sigma$ and $\Upsilon$ be two alphabets, a relation $R \subseteq \Sigma^* \times \Upsilon^*$ is said to be *finite-valued* just if for every $s \in \Sigma^*$, there are at most finitely many $u \in \Upsilon^*$ such that $Rsu$. If $M$ is a language over $\Upsilon$, define a language $R^{-1}[M]$ over $\Sigma$ by $R^{-1}[M] = \{s \mid \exists u(Rsu \wedge u \in M)\}$.

**Theorem 2.** *Let $\mathcal{M}$ be a class of languages over $\Upsilon$ that has finite elasticity, and let $R \subseteq \Sigma^* \times \Upsilon^*$ be a finite-valued relation. Then $\mathcal{L} = \{R^{-1}[M] \mid M \in \mathcal{M}\}$ also has finite elasticity.*

In order to obtain learnable subclasses of *B-NLC*, additional restrictions need to be imposed. Let $k$ be an upper bound on the number of occurences of any terminal, and let *k-B-NLC* denote the class of all *B-NLC* grammars with $k$ as such a bound. This bound implies a bound on the number of occurences of distinct non-terminal parts of daughter graphs. Since we assume a fixed alphabet and terminals in all rules, a bound on the number of rules in the grammar is implied, which implies a bound on the number of non-terminals.

**Proposition 3.** *For $k = 1$, $DL(\mathcal{G}_{k\text{-}B\text{-}NLC})$ has finite elasticity.*

*Proof.* Sketch: Assume that this class has infinite elasticity with trees $t_1, \ldots$ and derivation languages $D_1, \ldots$, with corresponding grammars $G_1, \ldots$. For any $i$, the set $G''_i$ of grammars in the class that generate a minimal derivation language and are consistent with $t_1 \ldots t_{i-1}$ is of finite cardinality. The grammar $G_i$ must be such that it is a superset of some such grammar, with a substitution applied to it. There are just a finite number of such substitutions for each $G'$, so after $p$ there can only occur a finite number of different grammars. Since $t_i \notin DL(G_i)$ and $\{t_1, \ldots t_{i-1}\} \subseteq DL(G_i)$, each of these grammars can only occur a finite number of times in the sequence. Thus, the whole sequence $G_1, \ldots$, and thus the whole sequence $D_1, \ldots$, must be of finite length. $\square$

Applying Theorem 2 twice, this result can be generalized to $k > 1$, and then from derivation- to graph language. It then follows that For any $k$, $GL(\mathcal{G}_{k\text{-}B\text{-}NLC})$ is learnable from positive data (graphs) by a consistent and conservative learner.

## References

[1] Cook, D.J., Holder, L.B. (eds.): Mining Graph Data. John Wiley & Sons, Chichester (2006)

[2] Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)

[3] Wright, K.: Identification of unions of languages drawn from an identifiable class. In: The 1989 Workshop on Computational Learning Theory, pp. 328–333. Morgan Kaufmann, San Mateo (1989)

[4] Motoki, T., Shinohara, T., Wright, K.: The correct definition of finite elasticity: Corrigendum to identification of unions. In: The Fourth Workshop on Computational Learning Theory. Morgan Kaufmann, San Mateo (1991)

[5] Kanazawa, M.: A note on language classes with finite elasticity. Technical Report CS-R9471, CWI, Amsterdam (1994)

# Inference of Uniquely Terminating EML

S. Kannamma[1], D.G. Thomas[2], and K. Rangarajan[2]

[1] S.D.N.B. Vaishnav College for Women, Chennai - 600 044
[2] Madras Christian College, Chennai - 600 059
dgthomasmcc@yahoo.com

## 1 Introduction

In [3], we have provided an algorithm to infer a few subclasses of linear languages through labeled extended Petri nets. The family of equal matrix languages [6] meets both the families of context sensitive languages and context-free languages. In this paper, we prove that an equal matrix language is a Petri net language. We construct labeled extended Petri nets to infer uniquely terminating code $k$-equal matrix languages (utCk-EMLs), a subclass of EMLs. A similar algorithm can be employed to construct a labeled Petri net which generates a uniquely terminating regular language [4]. This algorithm can be modified to infer a given uniquely terminating code regular language [2] through a labeled extended Petri net.

## 2 Algorithm to Infer Uniquely Terminating Languages

For the notions of a Petri net language and an equal matrix language we refer to [5,6]. A code $k$-equal matrix language and uniquely terminating code $k$-equal matrix language can be defined similar to code regular language [1] and uniquely terminating code regular language [2] respectively. We can show the following results.

**Theorem 1.** *An equal matrix language (EML) is a Petri net language.*

**Corollary 1.** *A code k-equal matrix language (Ck-EML) is a Petri net language.*

We present an algorithm to infer a uniquely terminating code $k$ - equal matrix language from positive data. This algorithm first develops a trie structure. It then gives rules to construct a labeled extended Petri net with its transitions labeled as code words over the given alphabet which generates the required language.

A set of sample words $\{w_1, w_2, \ldots, w_n\}$ from a uniquely terminating code $k$-equal matrix language and the code set $K$ are given as inputs to the inference algorithm. The following is the procedure to construct a labeled extended Petri net with the transitions labeled with code words from K generating the required utCk - EML.

**Algorithm PN - utCk - EML**

1. For each word in the sample do

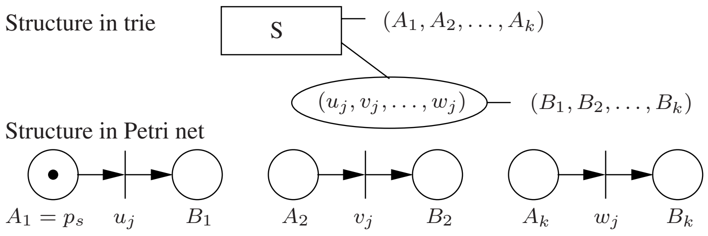
1a. Factorise the word using the code set $K$ and partition into $k$-equal parts as $x_i = \alpha_1\alpha_2\ldots\alpha_k$, $\alpha_i \in K^+$.

1b. If the $\alpha_i$'s are of the form $\alpha_1 = u_1u_2\ldots u_n$; $\alpha_2 = v_1v_2\ldots v_n$; $\ldots$; $\alpha_k = w_1w_2\ldots w_n$, form the $k$-tuples $(u_1, v_1, \ldots, w_1)$, $(u_2, v_2, \ldots, w_2)$, $\ldots$, $(u_n, v_n, \ldots, w_n)$ and store them as labels of nodes in the trie structure and label the root node as a $k$-tuple of nonterminals, say $(A_1, A_2, \ldots, A_k)$; $(u_n, v_n, \ldots, w_n)$, is the label of terminal node. Insert associated $k$-tuple nonterminals to the new nodes other than the terminal nodes.

1c. If there are nodes having children which are equally labeled final nodes, then merge the associated nonterminals of these nodes.

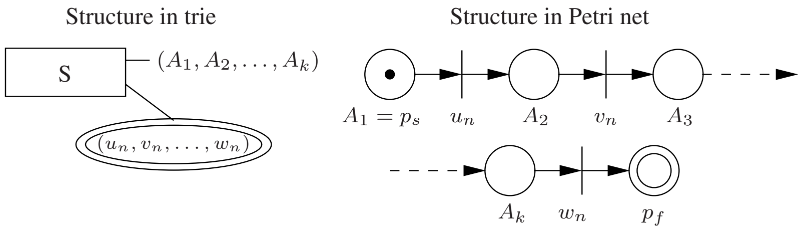
2. The construction of the resulting labeled extended Petri net is given as an output from the trie structure.

2a. If $S$ has a child, a node labeled by the $k$-tuple $(u_j, v_j, \ldots, w_j)$ with associated nonterminal $(B_1, B_2, \ldots, B_k)$, then make places $p_s = A_1$, $B_1$, $A_2$, $B_2$, $\ldots$, $A_k$, $B_k$ with a single token in $p_s$ and no tokens in other places and make transitions respectively labeled as $u_j, v_j, \ldots, w_j$ and the flow relation as shown below:

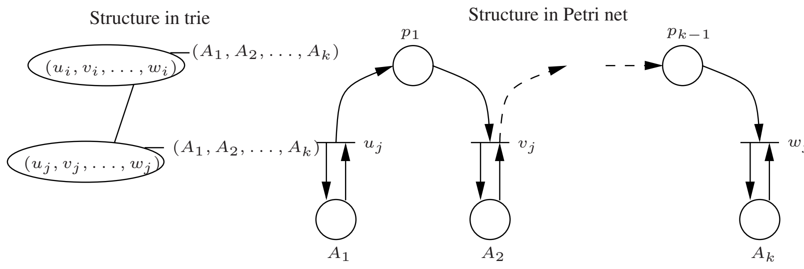


Remark: If $S$ is not the parent node, then, the procedure mentioned above holds good with the only condition that none of the places introduced have tokens in them.

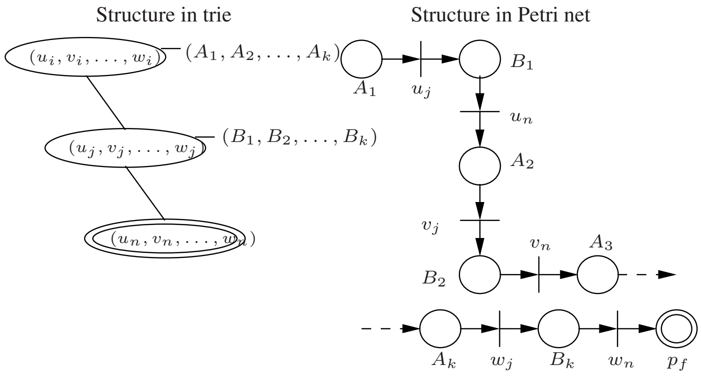
2b. If the node associated with $S$ has a child which is a final node labeled $(u_n, v_n, \ldots, w_n)$, then make places $p_s = A_1, A_2, \ldots, A_k$ and $p_f$, where, $p_f$ is a final place, with a single token in $p_s = A_1$ and no tokens elsewhere, as shown below:



2c. If a node labeled by the $k$-tuple $(u_j, v_j, \ldots, w_j)$ has a nonterminal $(A_1, A_2, \ldots, A_k)$ associated with it and if $(A_1, A_2, \ldots, A_k)$ is again the nonterminal associated with its parent, then make places $A_1, A_2, \ldots, A_k$ and $p_1, p_2, \ldots, p_{k-1}$; make transitions with labels as the words $u_j, v_j, \ldots, w_j$; the flow relation connecting these places and transitions is shown below:


Structure in trie
Structure in Petri net
$(u_i, v_i, \ldots, w_i)$
$(A_1, A_2, \ldots, A_k)$
$(u_j, v_j, \ldots, w_j)$
$(A_1, A_2, \ldots, A_k)$
$p_1$
$p_{k-1}$
$u_j$
$v_j$
$w_j$
$A_1$
$A_2$
$A_k$

2d. If the trie structure leading to a final node is a chain, the corresponding structure of the labeled extended Petri net is shown below:


Structure in trie
Structure in Petri net
$(u_i, v_i, \ldots, w_i)$
$(A_1, A_2, \ldots, A_k)$
$(u_j, v_j, \ldots, w_j)$
$(B_1, B_2, \ldots, B_k)$
$(u_n, v_n, \ldots, w_n)$
$A_1$
$u_j$
$B_1$
$u_n$
$A_2$
$v_j$
$v_n$
$A_3$
$B_2$
$A_k$
$w_j$
$B_k$
$w_n$
$p_f$

The initial marking of the constructed Petri net is with one token in $p_s$ and no tokens in other places and the final marking is with one token in $p_f$ and no tokens in other places. The language generated by this Petri net is the utCk - EML inferred in a sequence of conjectures.

## References

1. Emerald, J.D., Subramanian, K.G., Thomas, D.G.: Learning code regular and code linear languages. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 211–221. Springer, Heidelberg (1996)
2. Emerald, J.D., Subramanian, K.G., Thomas, D.G.: A note on inferring uniquely terminating code languages. Information Processing Letters 70, 217–222 (1999)
3. Kannamma, S., Thomas, D.G., Rangarajan, K.: On inference of uniquely terminating linear languages. In: Thangavel, K., Balasubramaniam, P. (eds.) Computing and Mathematical Modeling, pp. 291–298. Narosa Publishing House (2006)
4. Makinen, E.: Inferring uniquely terminating regular languages from positive data. Information Processing Letters 62, 57–60 (1997)
5. Peterson, J.L.: Petri net theory and modeling of systems. Prentice Hall, Englewood Cliffs (1981)
6. Siromoney, R.: On equal matrix languages. Information and control 14, 135–151 (1969)

# Estimating Graph Parameters Using Graph Grammars

Sourav Mukherjee[1] and Tim Oates[2]

[1] Department of Computer Science,
University of Maryland, Baltimore County, USA
Phone: +1-410-455-8790
sourav1@umbc.edu
[2] Department of Computer Science,
University of Maryland, Baltimore County, USA
oates@cs.umbc.edu

Stochastic graph grammars are probabilistic models suitable for modeling relational data, complex organic molecules, social networks, and various other data distributions [1]. In this paper, we demonstrate that such grammars can be used to reveal useful information about the underlying distribution. In particular, we demonstrate techniques for estimating the expected number of nodes, the expected number of edges, and the expected average node degree, in a graph sampled from the distribution. These estimation techniques use the underlying grammar, and hence do not require sampling. Experimental results indicate that our estimation techniques are reasonably accurate.

We use the notation $G = (V, E)$ to refer to a graph with $V$ being the set of vertices, and $E$ being the set of edges.

**Definition 1.** *Let $G = (V, E)$ be a graph. A* hyperedge *is an ordered subset of its vertices $V$. Alternatively, a hyperedge of degree $n$ can be thought of as a mapping $H : \{1, 2, ..., n\} \rightarrow V$. A* hypergraph *is a graph that can, in addition to edges, also have hyperedges.*

**Definition 2.** *A* (hyperedge replacement) stochastic context-free graph grammar *(SCFGG) is defined as a tuple $(S, N, T, P, p)$ where:*

- *$N$ is the set of non-terminal symbols,*
- *$T$ is the set of terminal symbols, disjoint from $N$,*
- *$S \in N$ is a special non-terminal called the start symbol,*
- *$P$ is a set of productions,*
- *$p$ is a probability function defined on the set of productions, such that the sum of the probabilities of all productions with the same left hand side equals 1.*

*In a hyperedge replacement SCFGG, terminals are used to denote graphs without hyperedges, while non-terminals are used to label hyperedges. A production is an ordered pair $(H, \alpha)$, written as $H \rightarrow \alpha$, where $H$ is a non-terminal and $\alpha$ is a hypergraph.*

A SCFGG can be viewed as a generative model: we start with the start symbol S, and at each step, we replace any non-terminal $H$ with a graph $\alpha$ such that there is a production $H \rightarrow \alpha$. This process is continued until we arrive at a graph that has no non-terminal symbols. When a hyperedge $H$ in a graph $G$ is replaced using the production $H \rightarrow \alpha$, $G$ is called the host-graph, and $\alpha$ is called the subgraph.

We now present techniques for estimating the expected number of vertices and nodes in a graph sampled from a distribution, given the grammar for the distribution. We define the following notation, which will aid the subsequent discussion. For any non-terminal $Z$, let us assume that there are $N_Z$ production rules with $Z$ on the left hand side, with probabilities $p_{Z,1}, p_{Z,2}, ..., p_{Z,N_Z}$, satisfying $\sum_{j=1}^{N_Z} p_{Z,j} = 1$. Let the $j^{th}$ such production be of the form $Z \rightarrow \alpha_j$ where $\alpha_j$ is a graph with $v_{Z,j}$ vertices, $a_{Z,j}$ edges and $h_{Z,j}$ hyper-edges, labeled $Z_{j,1}, Z_{j,2}, ..., Z_{j,h_{Z,j}}$. Note that these non-terminals do not have to be all distinct; they may even be the same as $Z$. Finally, let $D_Z$ denote the degree of the non-terminal $Z$.

For any non-terminal $Z$, let $n_Z$ represent the expected number of nodes in any graph obtained by expanding $Z$. Then the equation for $n_Z$ is given by:

$$n_Z = \sum_{j=1}^{N_Z} p_{Z,j}(v_{Z,j} + \sum_{k=1}^{h_{Z,j}} (n_{Z_{j,k}} - D_{Z_{j,k}})) \tag{1}$$

Thus we see that for each non-terminal $Z$ in the grammar, we will have a single linear equation, leading to a system to linear equations with the same number of equations as the number of non-terminals.

We now develop a system of linear equations for estimating the expected number of edges $e_Z$ in a graph, obtained from any non-terminal $Z$ in the grammar. The problem of estimating the expected number of edges is different from that of estimating the expected number of nodes, in that unlike nodes, edges are not glued together when a subgraph is embedded inside a host-graph. The equation for $e_Z$ is given by:

$$e_Z = \sum_{j=1}^{N_Z} p_{Z,j}(a_{Z,j} + \sum_{k=1}^{h_{Z,j}} e_{Z_{j,k}}) \tag{2}$$

Once again, we see that for each non-terminal $Z$ in the grammar, we will have a single linear equation, leading to a system of linear-equations with the same number of equations as the number of non-terminals.

Next we present two techniques, the *Naïve Degree Estimator* and the *Linear Degree Estimator* for estimating the average node degree of a graph generated from a given grammar. The average degree $\bar{d}$ of a node in a graph $G = (V, E)$ is defined as $\bar{d} = \frac{1}{|V|} \sum_{v \in V} d(v)$.We also know that $\bar{d} = \frac{2|E|}{|V|}$ . We will refer to this result as the Handshaking Lemma [2].

Given a non-terminal $Z$, let $\bar{d}_Z$ denote the expected value of the average degree of a node, of any graph obtained from $Z$. Then, we can estimate $\bar{d}_Z$ as:

$$\bar{d}_Z \approx \frac{2e_Z}{n_Z} \tag{3}$$

Of course, Equation 3 is only an approximate estimate, because the number of nodes and the number of edges are not, in general, independent. We now present a more accurate estimator.

Let, for a non-terminal $Z$, $\bar{d}_Z$ indicate the expected average node degree of any graph derived from the non-terminal symbol $Z$. Recall that the average is computed over all nodes in a graph, and the expectation is computed over the distribution of the graphs. Then, the expected number of nodes in the graph $\alpha_j$ is given by

$$n_{Z,j} = \sum_{k=1}^{h_{Z,j}} (n_{Z_{j,k}} - D_{Z_{j,k}}) + v_{Z,j} \tag{4}$$

Let us number the vertices in $\alpha_j$ as $1, 2, ..., v_{Z,j}$ and let for vertex $l (1 \leq l \leq v_{Z,j})$, $a_l$ be the number of terminal edges incident on that vertex. Then the expression for the expected average number of nodes is given by:

$$\bar{d}_Z - \sum_{j=1}^{N_Z} \sum_{k=1}^{h_{Z,j}} \frac{p_{Z,j}}{n_{Z,j}} \bar{d}_{Z_{j,k}} = \sum_{j=1}^{N_z} \sum_{l=1}^{v_{Z,j}} \frac{p_{Z,j}}{n_{Z,j}} a_l \tag{5}$$

Thus, we get a linear equation for every non-terminal $Z$ in the grammar. By solving this linear system, we can arrive at an estimate of the expected average node degree.

Graph grammars are useful probabilistic models for distributions over graphs because they are compact, hierarchical, and amenable to interpretation by domain experts. However, in this paper, we have demonstrated that the utility of graph grammars goes beyond elucidation of structure and generation of samples. We have presented grammar-based techniques to estimate the expected number of nodes, the expected number of edges, and the expected average node degree in a graph generated by the grammar. We have also presented a characterization of grammars that can produce graphs that are not connected. Future directions include exploring the characterization of grammars the generate planar graphs, and applying these results to real-life domains such as relational databases, organic molecules, and social networks.

## References

1. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1. World Scientific, Singapore (1997)
2. West, D.B.: Introduction to Graph Theory, 2nd edn. Prenctice Hall, Upper Saddle River (2001)

# Learning of Regular $\omega$-Tree Languages

M. Jayasrirani[1], M.H. Begam[1], and D.G. Thomas[2]

[1] Arignar Anna Government Arts College, Walajapet
[2] Madras Christian College, Chennai - 600 059
dgthomasmcc@yahoo.com

**Abstract.** We introduce two subclasses of regular $\omega$-tree languages called local $\omega$-tree languages and Buchi local $\omega$-tree languages. Automata characterization for these $\omega$-tree languages is given. For these subclasses and $\omega$-regular tree languages learning algorithms are given.

## 1 Introduction

The theory of tree automata and tree languages emerged in the middle of 1960s. Saoudi et al. [2] have considered infinite trees ($\omega$-trees), recognizable $\omega$-tree languages and regular $\omega$-tree languages. Infinite trees are useful to decide second order theories. In this paper local $\omega$-tree languages and Buchi local $\omega$-tree languages are defined and automata characterization for $\omega$-regular tree languages in terms of local $\omega$-tree languages and Buchi local $\omega$-tree languages is given. There is no learning algorithm so far in the literature for the local $\omega$-tree languages, Buchi local $\omega$-tree languages and regular $\omega$-tree languages. We give learning algorithms for these classes of $\omega$-tree languages. Our approach is similar to the one given in [3].

## 2 Definitions and Results

Definitions concerning trees, root of a tree, frontier of a tree, forks of a tree, infinite trees, automata on infinite trees and ultimately periodic infinite trees can be found in [1,2].
$T_\Sigma$ stands for the set of all finite trees over $\Sigma$.
$T_\Sigma^\omega$ stands for the set of all infinite trees over $\Sigma$.
$root(t)$ stands for root of a tree $t$.
$fork(t)$ stands for fork of a tree $t$.
$fork(\Sigma)$ stands for the set of all forks of $\Sigma$-trees.
$Frfork(t)$ stands for the set of all forks of a tree $t$ that end with frontiers of $t$.

**Definition 1.** *A $\omega$-tree language $L \subseteq T_\Sigma^\omega$ is called a local $\omega$-tree language if there exists a pair $S = \{R, E\}$ (called a local system) where $R \subseteq \Sigma$ and $E \subseteq fork(\Sigma)$ such that*

$$L = \{t \in T_\Sigma^\omega : root(t) \in R, fork(t) \subseteq E\}$$

*The elements in $fork(\Sigma)$ occur infinitely many times. In this case we write $L = L^\omega(R, E)$. The set of all local $\omega$-tree languages is denoted by $\mathcal{L}^\omega$.*
*$L = \{a(b^\omega, c^\omega), a(c^\omega, b^\omega)\}$ is a local $\omega$-tree language.*

**Definition 2.** *A Buchi local system over $\Sigma$ is an ordered triple $S = \{R, E, E'\}$ where $R \subseteq \Sigma$, $E \subseteq fork(\Sigma)$ and $E' \subseteq E$. We denote $L^{\omega}(R, E, E')$ a Buchi local $\omega$-tree language defined as*

$$L'(R, E, E') = \{t \in T_{\Sigma} : root(t) \in R, fork(t) \subseteq E,\ inf\,fork(t) \cap E' \neq \phi\}$$

*where $inf\,fork(t)$ is the set of elements in $fork(t)$ which occur infinitely many times in $t$. An $\omega$-tree language $L \subseteq T_{\Sigma}^{\omega}$ is called a Buchi local $\omega$-tree language if there exists a Buchi local system such that $L = L^{\omega}(R, E, E')$. The set of all Buchi local $\omega$-tree languages is denoted by $\mathcal{L}_{BE}^{\omega}$. $L = \{a(b^{\omega}, c^{\omega}), a(c^{\omega}, b^{\omega})\}$ is a Buchi local $\omega$-tree language.*

**Theorem 1.** *Every regular $\omega$-tree language (recognizable $\omega$-tree language) is an alphabetic homomorphic image of a Buchi local (local) $\omega$-tree language.*

We can give construction procedures for deterministic Buchi $k$-ary $\omega$-tree automaton $M$ such that $L = L^{\omega}(M)$ where $L$ is a local (Buchi local) $\omega$-tree language.

## 3 Learning Buchi Local $\omega$-Tree Languages

**Definition 3.** *Let $L \in \mathcal{L}_{BE}^{\omega}$ be such that $L = L^{\omega}(S)$ for some Buchi local system $S = (R, E, E')$ over an alphabet $\Sigma$. $S$ is said to be minimal for $L$, if for any other Buchi local system $S_1 = (R_1, E_1, E_1')$ over $\Sigma$ with $L = L^{\omega}(S_1)$ we have $R \subseteq R_1$, $E \subseteq E_1$ and $E' \subseteq E_1'$.*

**Definition 4.** *Let $K$ be a finite sample of ultimately periodic infinite trees. Let $R_K = root(K) = \{root(t) : t \in K\}$, $E_K = fork(K) = \cup_{t \in K} fork(t)$*

$$E_K' = \cup_{a(b^{\omega}, c^{\omega})} F_r fork(t)$$

*$S_K = (R_K, E_K, E_K')$ is called a Buchi local system associated with $K$ and $L = L^{\omega}(S_K)$ is called Buchi local $\omega$-tree language associated with $K$.*

**Theorem 2.** *If $K, K'$ are finite samples of ultimately periodic $\omega$-trees of $T_{\Sigma}^{\omega}$ then*

1. *$K \subseteq L^{\omega}(S_K)$*
2. *$K \subseteq K'$ implies $L^{\omega}(S_K) \subseteq L^{\omega}(S_{K'})$*
3. *$L \in \mathcal{L}_{BE}^{\omega}$ with $K \subseteq L$ implies $L^{\omega}(S_K) \subseteq L$*

**Definition 5.** *Let $L$ be a local (Buchi local) $\omega$-tree language. A finite subset $F$ of $T_{\Sigma}^{\omega}$ is called a characteristic sample for $L$ if $L$ is the smallest local (Buchi local) $\omega$-tree language containing $F$.*

**Theorem 3.** *If $F$ is the characteristic sample for a local (Buchi local) $\omega$-tree language and $F \subseteq K \subseteq L$ then $L = L^{\omega}(S_K)$.*

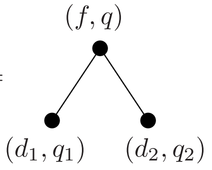
**Theorem 4.** *There effectively exists a characteristic sample for any local (Buchi local) $\omega$-tree language.*

**Theorem 5.** *Given an unknown local (Buchi local) $\omega$-tree language we give an algorithm that learns in the limit from positive data, a local system (Buchi local system) $S_F$ such that $L^{\omega}(S_F) = L$.*

## 4 Learning Regular $\omega$-Tree Languages

In this section we give a learning algorithm for regular $\omega$-tree languages from positive data and restricted superset queries.

If $L$ is a regular $\omega$-tree language over $\Sigma$ and if $L$ is recognized by a Buchi $k$-ary $\omega$-tree automaton $M =< Q, \Sigma, \delta, q_0, F >$ then by theorem 1 there is a Buchi local $\omega$-tree language $U$ over $\Omega$ and a strictly alphabetic morphism $h : \Omega \rightarrow \Sigma$ such that $h(U) = L$. Consider an ultimately periodic infinite tree $a(b^\omega, c^\omega)$ in $U$. Let $t = a(b(b, b), c(c, c))$ where $a, b, c \in \Sigma$. Let $g(t) = (f, q) < (d_1, q_1), (d_2, q_2) >$ where $d_i =$ root of $(t_i)$ $(i = 1, 2, t_i$ are the subtrees of $t)$ be a tree over $\Omega$. i.e.,

$g(t) =$ 

The tree $g(t)$ is said to be a good tree for $t$ if $d_1, d_2$ are the children of $f$. Let $G(t)$ be the set of all good trees in $h^{-1}(t)$ for $t$. If $H_U$ is a characteristic sample for $U$, then there exists a finite set of positive data $S_L$ of $L$ such that $H_U \subseteq h^{-1}(S_L)$.

We provide a learning algorithm for regular $\omega$-tree languages from positive data and restricted superset queries.

**Theorem 6.** *Given an unknown regular (recognizable) $\omega$-tree language $L$, we can give algorithm that effectively learns from positive data and restricted superset queries, a Buchi local (local) system $S$ such that $L = h(L^\omega(S))$.*

## References

1. Gecseg, F., Steinby, M.: Tree languages. In: Handbook of Formal Languages, vol. 3, pp. 1–68. Springer, Heidelberg (1997)
2. Saoudi, A.: Rational and recognizable infinite tree sets. In: Tree Automata and Languages, pp. 225–234. Elsevier Science, Amsterdam (1992)
3. Saoudi, A., Yokomori, T.: Learning local and recognizable $\omega$-languages and Monadic Logic Programs. In: Proc. EuroColt, 1993, pp. 157–169. Oxford Univ. Press, Oxford (1994)

# Inducing Regular Languages Using Grammar-Based Classifier System

Olgierd Unold

Institute of Computer Engineering, Control and Robotics,
Wroclaw University of Technology, Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland
olgierd.unold@pwr.wroc.pl
http://olgierd.unold.staff.iiar.pwr.wroc.pl/

**Abstract.** This paper takes up the topic of a task of training Grammar-based Classifier System (GCS) to regular grammars from data. GCS is a new model of Learning Classifier Systems in which the population of classifiers has a form of a context-free grammar rule set in a Chomsky Normal Form. Near-optimal solutions or better than reported in the literature were obtained.

**Keywords:** Grammatical Inference, Learning Classifier Systems, FSA Induction.

## 1 Introduction

In this paper, we are interested in inducing a grammar that accepts a regular language (RL) given a finite number of positive and negative examples drawn from that language. The approaches to learning RL or equivalent Deterministic Finite Automata (DFA) base mainly on evolutionary algorithms, recurrent neural network or combination of these two methods. It has been proved that RL/DFA induction is a hard task by a number of criteria.

This paper addresses RL induction using Grammar-based Classifier System (GCS) [5] - a new model of Learning Classifier System (LCS). LCS is machine learning paradigm introduced by Holland [2]. It exploits evolutionary computation and reinforcement learning to develop a set of condition-action rules (the classifiers) which represent a target task that the system has learned from online experience. Although there are some approaches to handle with context-free grammar (CFG), there is no one work on inducing RLs with LCSs.

GCS [5] evolves population of classifiers in a form of a CFG rule set, each rule in a Chomsky Normal Form (CNF). CNF allows only production rules in the form of $A \rightarrow a$ or $A \rightarrow BC$, where $A$, $B$, $C$ are the non-terminal symbols and $a$ is a terminal symbol. The first rule is an instance of terminal rewriting rule not affected by the genetic algorithm (GA), and generated automatically as the system meets unknown terminal symbol. Left hand side of the rule plays a role of classifier's action while the right side a classifier's condition. All classifiers form a population (one CFG) of evolving individuals. Environment of classifier system is made up by an array of Cocke-Younger-Kasami parser. GCS matches the rules according to the current environmental state (state of parsing) and generates an action/actions pushing the parsing process toward the complete derivation of the sentence analyzed. Apart from the GA, the *covering* procedure explores the space searching for new, better productions. We refer the reader to [5] for more details of GCS.

## 2 Regular Language Induction Using GCS

The datasets most commonly used in DFA learning is Tomita sets. In this paper GCS will be compared against the evolutionary methods proposed by Lucas and Reynolds [3] and Luke et al. [4]. Both methods present one of the best-known results in the area of RL/DFA induction. All of compared evolutionary methods will assume the same training and test sets. Some comparisions will be made also to EDSM method [1], the current most powerful passive approach to DFAs inference. Fifty independent experiments were performed, evolution on each training corpus ran for 5,000 generations, with the following GCS parameters: number of nonterminal symbols 19, number of terminal symbols 7, crossover probability 0.2, mutation probability 0.8, population consisted of maximal 40 classifiers where 30 of them were created randomly in the first generation, crowding factor 18, crowding size 3. The approach presented in [4] (denoted by GP) applies gene regulation to evolve DFA. In this approach genes are states in the automaton, and a gene-regulation-like mechanism determines state transitions. Each gene has Boolean value indicating whether or not it was an accepting state. The main results are summarized in Table 1. For compared methods induction of L3 language appeared to be hard task. Both in GP and in GCS only the one run over 50 successfully finished. But GP found the solution in 12450 iterations, whereas GCS in only 666 steps. For the same language GCS correctly classified all of the unseen examples, while GP achieved 66%. As to an indicator nGen, GP was not able correctly classified unseen strings for any language from the tested corpora, while GCS induced a grammar fully general to the language in 4 cases. It is interesting to compare the results of induction for L5 language. GP approach could not find the proper grammar (DFA) for any run, while GCS found the solution in all runs, on average in 201 steps. While learning L1 and L2 languages, GP found the proper grammars not in all runs, whereas for GCS this task appeared to be trivial (100% nGen, 50/50 nSuccess, and nEvals 2 steps) Table 1 shows also the cost of induction (an indicator nEvals) for the methods Plain, Smart (Sm), and nSmart (nSm) taken from [3], GP approach, and GCS. Lucas i Reynolds [3] used different method to evolving DFA. In contrary to [4], only transition matrix was evolved, supported by a simple deterministic procedure to optimally assign state labels. This approach is based on evolutionary strategy (1+1). Three versions of induction algorithm were prepared: an approach in which both the transition matrix and the state label vector evolve (Plain), so-called Smart method evolving only the transition matrix and the number of the states was fixed and equal to 10, and finally nSmart method in which the number of the DFA states is equal to the size of minimal automata. GCS obtained the best results for the

**Table 1.** Comparison of GCS with different evolutionary methods and non-evolutionary EDSM. For each learning corpus, the table shows the target language, and three sets of results. The first indicator nSuccess is the number of runs with success gained by GCS within 50 experiments and compared approach. The second one nGen is the percentage of all unseen strings correctly classified, and the last one nEvals indicates the average number of generations needed to reach the 100% fitness.

| **Lang.** | **nSuccess** | | **nGen** | | | | | **nEvals** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **GP** | **GCS** | **GP** | **GCS** | **Sm** | **nSm** | **EDSM** | **GP** | **GCS** | **Plain** | **Sm** | **nSm** |
| L1 | 31/50 | 50/50 | 88.4 | 100 | 81.8 | 100 | 52.4 | 30 | 2 | 107 | 25 | 15 |
| L2 | 7/50 | 50/50 | 84.0 | 100 | 88.8 | 95.5 | 91.8 | 1010 | 2 | 186 | 37 | 40 |
| L3 | 1/50 | 1/50 | 66.3 | 100 | 71.8 | 90.8 | 86.1 | 12450 | 666 | 1809 | 237 | 833 |
| L4 | 3/50 | 24/50 | 65.3 | 100 | 61.1 | 100 | 100 | 7870 | 2455 | 1453 | 177 | 654 |
| L5 | 0/50 | 50/50 | 68.7 | 92.4 | 65.9 | 100 | 100 | 13670 | 201 | 1059 | 195 | 734 |
| L6 | 47/50 | 49/50 | 95.9 | 96.9 | 61.9 | 100 | 100 | 2580 | 1471 | 734 | 93 | 82 |
| L7 | 1/50 | 11/50 | 67.7 | 92.0 | 62.6 | 82.9 | 71.9 | 11320 | 2902 | 1243 | 188 | 1377 |

L1 and L2 languages among comparable methods. The result 201 steps for L5 is comparable with the best result of 195 reached by nSmart. Although GCS reached similar result for language L3 as the best method (666 for GCS, and 237 for Smart), it is hard to compare for this language these methods, because of low value of nSuccess for GCS - only one run over 50 finished with success. For the languages L4, L6, and L7 fixed-size structured methods (Plain, Smart, and nSmart) achieved better results than variable-size methods (GP and GCS). Finally, Table 1 shows the percentage of all unseen strings correctly classified (an indicator nGen) for the methods Smart, nSmart, EDSM, GP, and GCS. Recall that the EDSM, as a heuristic and non-evolutionary method, was single-time executed during learning phase. Model GCS achieved the best results from all tested approaches for L1, L2, L3, and L7 languages. For the language L4 the same 100% accuracy was obtained by proposed method, nSmart, and EDSM. For the L5 and L6 languages GCS obtained the second result, higher than 90%.

## References

1. Cicchello, O., Kremer, S.C.: Beyond EDSM. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 37–48. Springer, Heidelberg (2002)
2. Holland, J.: Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Michalski, R.S., et al. (eds.) Machine Learning, an Artificial Intelligence Approach, vol. II, pp. 593–623. Morgan Kaufmann, San Francisco (1986)
3. Lucas, S., Reynolds, T.J.: Learning DFA: Evolution versus Evidence Driven State Merging. In: Proc. Congress Evolutionary Computation, pp. 351–358 (2003)
4. Luke, S., Hamahashi, S., Kitano, H.: 'Genetic' Programming. In: Banzhaf, W., et al. (eds.) Proc. Genetic and Evolutionary Computation Conf., pp. 1098–1105 (1999)
5. Unold, O.: Context-free grammar induction with grammar-based classifier system. Archives of Control Science 15 (LI)(4), 681–690 (2005)

# Problems with Evaluation of Unsupervised Empirical Grammatical Inference Systems

Menno van Zaanen and Jeroen Geertzen

Dept. of Communication & Information Sciences
Tilburg University
Tilburg, The Netherlands
{mvzaanen,j.geertzen}@uvt.nl

**Abstract.** Empirical grammatical inference systems are practical systems that learn structure from sequences, in contrast to theoretical grammatical inference systems, which prove learnability of certain classes of grammars. All current empirical grammatical inference evaluation methods are problematic, i.e. dependency on language experts, appropriateness and quality of an underlying grammar of the data, and influence of the parameters of the evaluation metrics. Here, we propose a modification of an evaluation method to reduce the ambiguity of results.

## 1 Introduction

Grammatical inference (GI) can be described as the inference or induction of structure from sequences of symbols. We distinguish three sub-fields in the field of grammatical inference: formal GI, empirical GI, and applied GI [1].

Formal GI investigates which classes of grammars can be learned within certain bounds of algorithmic complexity and gives mathematical proofs for this. Empirical GI develops practical systems learning grammars. Often, the underlying (class of the) grammar is unknown. Applied GI is a collection of research that explores or employs GI as a step towards another research goal.

Here, we will review the evaluation methods that are available for measuring the performance of *empirical* GI systems. The sub-field of empirical GI does not allow formal proofs and allows for generic evaluation techniques.

## 2 Current Evaluation Approaches

Evaluation of GI systems is carried out by applying the system to unstructured data, and evaluating its output. The different methods can be divided into four groups: *looks-good-to-me* approaches analyze the output of GI systems manually. *Rebuilding a-priori known grammars* use, often small, "toy" grammars to generate sequences, which are used as input for the GI system. The output of the system is then compared against the original grammar. The *language membership* method measures the ability to classify sequences based on language membership. This measures language equivalence (weak equivalence). The performance in this method is expressed by two metrics: *precision*, which showsthe effectiveness to decide whether a sequence is in the language or not and *recall*, which measures coverage. Finally, *comparison against a treebank* uses a treebank, a collection of sequences with their derivation, as a "gold standard". The plain sequences (generated by removing the structure from the treebank sequences) are used as input and the output of the GI system is compared against the original structure. [2, 3]

## 3 Problems with Current Approaches

All evaluation methods have their problems. The *looks-good-to-me* approach is highly subjective. Evaluation performed by the GI system designer is biased and even for external experts it is hard to maintain consistency between systems.

*Rebuilding known grammars* resolves the dependency on experts and biased results, but only small grammars can be tested and scalability is not taken into account. Also, the grammars can be tuned to generate positive results.

The *language membership* methods depend heavily on several design choices. There are different ways to select negative sequences, which has an impact on the results. Similarly, the recall metric requires a sequence generation method, which may also have a large influence.

The *compare against treebank* approach is unbiased with respect to the evaluator and is scalable. However, it still has settings that have a significant impact. This will be discussed in the next section.

The *compare against treebank* and *language membership* methods have most potential. However, the problems of the *language membership* approach require more research, so we will concentrate on the *compare against treebank* approach.

## 4 Evaluating Compare against Treebank

The *compare against treebank* method uses precision (correctness) and recall (completeness) on tree structures (PARSEVAL [4]) as metrics.

The learned structure is compared against the gold standard, which may contain "trivial" structure. Examples are structures spanning the entire sequence or only a single word. This structure has a impact on the evaluation.

We applied the Alignment-Based Learning system [2] to the ATIS treebank, taken from the Penn Treebank [5] and varied the amount of trivial structure to get an indication of its impact on the evaluation scores. The first column in Table 1 shows the scores on all structure. Columns that are marked with $-e$ discard empty brackets, $-s$ discards brackets spanning the full sentence, and $-w$ discards spans containing a single word only.

Both the micro-average, where counts of correct brackets over the entire treebank are collated, and the macro-average scores are calculated. Micro-averaging is better in showing actual performance by taking bracket distribution per sentence into account.

The difference between macro- and micro-averaging is substantial and there are major differences with varying amounts of trivial structure. We propose to

**Table 1.** Results of Alignment-Based Learning using different evaluation parameters

| | | -e | -s | -w | -e-s | -e-w | -s-w | -e-s-w |
|---|---|---|---|---|---|---|---|---|
| Macro Recall | 56.18 | 56.18 | 55.73 | 49.19 | 55.73 | 49.19 | 46.79 | 46.79 |
| Macro Precision | 51.13 | 80.57 | 51.07 | 26.26 | 81.75 | 58.24 | 25.40 | 58.57 |
| Macro F-Score | 53.53 | 66.20 | 53.30 | 34.24 | 66.28 | 53.33 | 32.92 | 52.02 |
| Micro Recall | 49.31 | 49.31 | 49.00 | 40.67 | 49.00 | 40.67 | 39.69 | 39.69 |
| Micro Precision | 51.00 | 79.67 | 51.15 | 25.27 | 80.61 | 56.13 | 25.05 | 57.22 |
| Micro F-Score | 50.14 | 60.91 | 50.05 | 31.17 | 60.95 | 47.17 | 30.72 | 46.87 |

use the micro-averaged PARSEVAL metrics without trivial structure. This is the most strict evaluation, which, in this case, results in an F-score of 46.87.

## 5 Conclusion

We reviewed empirical GI evaluation approaches, which all have problems. The *compare against treebank* approach is most promising, but it is essential to define the exact settings of the evaluation as they have a major impact in the actual results. We propose to remove all trivial structure and use micro-averaged PARSEVAL metrics. Most published results are difficult to compare and interpret, because the exact evaluation settings are unknown.

## References

[1] Adriaans, P.W., van Zaanen, M.M.: Computational grammatical inference. In: Holmes, D.E., Jain, L.C. (eds.) Innovations in Machine Learning. Studies in Fuzziness and Soft Computing, ch. 7, vol. 194. Springer, Berlin (2006)

[2] van Zaanen, M.: Bootstrapping Structure into Language: Alignment-Based Learning. PhD thesis, University of Leeds, Leeds, UK (January 2002)

[3] van Zaanen, M., Roberts, A., Atwell, E.: A multilingual parallel parsed corpus as gold standard for grammatical inference evaluation. In: Kranias, L., Calzolari, N., Thurmair, G., Wilks, Y., Hovy, E., Magnusdottir, G., Samiotou, A., Choukri, K. (eds.) Proceedings of the Workshop: The Amazing Utility of Parallel and Comparable Corpora, Lisbon, Portugal, pp. 58–61 (May 2004)

[4] Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., Strzalkowski, T.: A procedure for quantitatively comparing the syntactic coverage of English grammars. In: Proceedings of a Workshop—Speech and Natural Language, February 19–22, pp. 306–311 (1991)

[5] Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a large annotated corpus of English: the Penn treebank. Computational Linguistics 19(2), 313–330 (1993)

# Author Index