INFERRING REGULAR LANGUAGES BY MERGING NONTERMINALS

Erkki Mäkinen

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1997-6

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1997-6, MAY 1997

INFERRING REGULAR LANGUAGES BY MERGING NONTERMINALS

Erkki Mäkinen

University of Tampere
Department of Computer Science
P.O.Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4170-2
ISSN 0783-6910

Inferring regular languages by merging nonterminals

Erkki Mäkinen

Department of Computer Science, University of Tampere, P.O. Box 607, FIN-33101 Tampere, Finland

Abstract

Several subclasses of regular languages are known to be inferable from positive data only. This paper surveys classes of languages originating from the class of reversible languages. We define the classes by using a uniform grammatical notation.

Keywords: regular language, grammatical inference, identification in the limit, merging.

1 Introduction

Gold [9] has shown that regular languages cannot be inferred from positive data only. This negative result has initiated a search for subclasses of regular languages having the desirable inference property. From the viewpoint of the present paper, the class of k-reversible languages introduced by Angluin [3] was the first interesting subclass of regular languages proved to be inferable from positive data only. Later, several other such subclasses have been found.

The purpose of this paper is to survey certain subclasses of regular languages infereable from posivite data. We restrict ourselves to classes of languages whose definition resembles that of k-reversible languages. These classes include at least the following: k-contextual languages [15], Szilard languages of regular grammars [12], strictly regular languages [18,20], (k, h)-contextual languages [1], code regular languages [6], and uniquely terminating regular languages [14]. On the other hand, we exclude such notable classes of languages as regular pattern languages [2] and paranthesis languages [5] since we consider them to be outside the "family of language classes" originating from reversible languages.

Some of the language classes of our interest have been originally defined by using automata concepts and some others by using grammar concepts. Our main idea is to define all the classes of languages considered here in a uniform manner. We have chosen to use grammatical definitions. This should not cause any confusion, since the correspondence between regular grammars and finite automata is obvious (although a care must take when fixing some special features). We hope that the uniform treatment of the classes of language will make it easier to find new relationships between them.

2 Preliminaries

We assume a familiarity with the basics of formal language theory and grammatical inference as given e.g. in [10] and [4], respectively. If not otherwise stated, we follow the notations and definitions given in these references.

We consider regular grammars only. Following [10], we denote regular grammars as $G = (V, \Sigma, P, S)$, where Σ is the set of terminals and V is the union of Σ and the set N of nonterminals. As always, P and S are the set of productions and the start symbol, respectively. The length of a string w is denoted by $lg(w)$, and the left-quotient of L and w by $T_L(w) = \{v \mid wv \in L\}$.

The productions having A in their left hand side are called *A-productions*. A production of the form $A \to b$, where b is a terminal, is said to be *terminating*; otherwise a production is said to be *continuing*. A continuing production has the form $A \to bB$, where b is a terminal and B is a nonterminal. Other forms of productions are not allowed.

Notice that for notational simplicity, we do not allow λ-productions (productions with the empty string λ in the right hand side). Similarly, since the start symbol in a grammar is unique, we read the earlier definitions given in automata formalism as if the set of initial states were always a singleton set.

In the grammatical inference literature it is a common practice that the final states of finite automata can have outgoing transitions. If $\delta(q, a) = q_f$ is a transition to a final state q_f having outgoing transitions, then the corresponding regular grammar must contain both the terminating production $A_q \to a$ and the continuing production $A_q \to aA_{q_f}$ where A_q and A_{q_f} are the nonterminals corresponding to the states q and q_f, respectively. This follows from our earlier decision not to allow λ-productions.

A finite automaton is said to be *deterministic* if, for each state, the leaving transitions have unique labels. Similarly, a regular grammar is usually considered to be deterministic if, for each nonterminal A, the right hand sides of all A-productions begin with unique terminals. However, to keep the automata and grammar notations consistent, we have to allow productions $A \to a$ and $A \to aB$ in a deterministic regular grammar.

The basic operation in our inference algorithms is *merging*. If automata formalism is used, the inference algorithms merge states in finite automata. Since we use grammar notation, the inference algorithms to be described merge nonterminals instead of states. Merging a pair A and B of nonterminals simply means that all appearances of B are replaced by A's (the roles of A and B are arbitrary). Suppose that we have productions $C \to aA$ and $C \to a$. Then A corresponds to a final state of the underlying finite automaton. When A and B are merged, it must be possible that also the derivation using B can be terminated here. Hence, for each production $D \to bB$, we first replace B by A, and further, we add a new production $D \to b$ to the grammar (if it is not already there).

As inference criterion we use "identification in the limit" [9]. In our case the conjectures outputted by the inference algorithms are regular grammars. After obtaining the ith input word, an inference algorithm outputs a conjecture G_i. Inference in the limit means that there is an algorithm outputting, for any positive representation of the language L to be inferred and for sufficiently large i, the correct grammar G_i generating L, and not changing the conjecture after reading further input words.

In the grammatical inference literature the general treatment of a class of languages can contain e.g. the following tasks:

- prove that an inference algorithm exists
- prove that the inference algorithm outputs the next conjecture in polynomial time
- prove that characteristic samples exist (a characteristic sample of a language L is a positive sample S_0 if L is the smallest language in the given class of languages containing S_0)
- prove that the algorithms output the smallest language containing the sample in the desirable class of languages.

The whole repertory is proved for some classes of languages (see [3,20,1]). Our purpose here is not to complete the list for other classes.

The efficiency of an inference algorithm is measured by the time complexity of outputting the next conjecture, and by the number of *implicit errors of prediction* made. An inference algorithm is said to make an implicit error of prediction at the ith step if the conjecture G_i fails to produce the $(i+1)$th input word [16].

3 Szilard languages of regular grammars

Instead of using the chronological order (and starting with the reversible languages), we start by introducing the most rudimentary of the present language classes, the Szilard languages of regular grammars. This class of languages is contained in all the classes to be introduced in the forthcoming sections, except in the class of uniquely terminating regular languages to be discussed in section 6.

Let $G = (V, \Sigma, P, S)$ be a regular grammar whose productions are uniquely labeled by the symbols of an alphabet C. If a production $A \to \alpha$ is associated with the label a we write $a : A \to \alpha$. If a sequence w of labeled productions is applied in a derivation $\beta \Rightarrow^* \gamma$, we write $\beta \Rightarrow^w \gamma$. The Szilard language $Sz(G)$ of G is defined as (see [11])

$$Sz(G) = \{w \in C^+ \mid S \Rightarrow^w x, x \in \Sigma^*\}.$$

The class of Szilard languages of regular grammars is denoted by $\mathcal{SZ}$. Languages in $\mathcal{SZ}$ can be generated by regular grammars with unique terminals in the right hand sides of the productions.

The inference algorithm for Szilard languages of regular grammars — and also for the other classes of languages considered here — is simple: Apply merges as long as the defining conditions of the class of languages in question are violated. When no merges are possible, the desired language is obtained.

The inference algorithm for $\mathcal{SZ}$ makes use of the following properties of Szilard languages:

(i) If a word in $Sz(G)$ begins with a symbol a then the production producing a has the start symbol S in its left hand side.
(ii) Each word in $Sz(G)$ ends with a symbol produced by a terminating production.
(iii) An appearance of a pair of consecutive symbols $a_i a_j$ in a word in $Sz(G)$ implies that the corresponding nonterminals A_i and A_j and the production $A_i \to a_i A_j$ are used in the derivation.

Items (i) and (ii) above are common to all classes of languages considered in this paper. Item (iii) is specific to $\mathcal{SZ}$. Szilard languages have the special feature that in (i) and (ii) the productions are unique.

We first suppose that the grammar to be inferred has the same number of nonterminals as it has productions and that the left hand side of the production having a_i in its right hand side is the nonterminal A_i, $i = 1, \ldots, n$. The algorithm merges all such pairs of nonterminals A_i and A_j for which we have a common predecessor of a_i and a_j in the sample. Namely, all sentential forms in regular grammars contain at most one nonterminal, and if we are able to apply two different productions immediately after the same production the two applicable productions must have identical left hand sides.

Notice that we already know that all appearances of a certain terminal are produced by the same production. The the most general assumption to start with is then "the same number of nonterminals as productions". (The number of terminals and prodcutions coincide.) In the other inference algorithms to be discussed later in this paper we do not know whether all appearances of a terminal are produced by the same production. It follows that we have to start with weaker assumptions than "the same number of nonterminals as productions". Usually, we have to start with the assumption "the same number of nonterminals as appearances of terminals". The conditions for performing a merge operation naturally depend on the specific features of the class of languages in question, and they differ from the condition (ii) above.

Given a finite sample of positive data, the problem of finding a language in $\mathcal{SZ}$ compatible with the sample can be solved in time $\mathcal{O}(t)$, where t is the total length of the words in the sample [12]. This follows essentially from the fact that we can use the off-line version of the well-known set union algorithm [7] when maintaing the nonterminal sets manipulated by the merges. For details concerning the set union algorithms, see [19].

Example 1 *Consider a sample $\{a_1a_2a_2a_2a_3, a_1a_2a_4\}$ from a Szilard language of a regular grammar. We start with a regular grammar having the productions $A_1 \to a_1A_2$, $A_2 \to a_2A_2$, $A_2 \to a_2A_3$, and $A_3 \to a_3$ (by $a_1a_2a_2a_2a_3$) and $A_1 \to a_1A_2$, $A_2 \to a_2A_4$, and $A_4 \to a_4$ (by $a_1a_2a_4$). As a_2 is a common predecessor of a_2 and a_3 we must have $A_2 = A_3$. Moreover, a_3 and a_4 have a common predecessor a_2. After merging $A_2 = A_3 = A_4$, we obtain the resulting grammar with the productions $A_1 \to a_1A_2$, $A_2 \to a_2A_2$, $A_2 \to a_3$, and $A_2 \to a_4$, where A_1 is the start symbol.*

Szilard languages of regular languages are polynomial-time inferable from positive data in a certain strong sense defined by [16]: Not only is it possible to output the next conjecture in polynomial time on the total length of the input words so far read, but also the number of implicit errors of prediction made is polynomial on the size of the final "right" conjecture [13]. As far as we know, $\mathcal{SZ}$ is the only non-trivial class of languages which is proved to be polynomial-time inferable from positive data in Pitt's sense.

Yokomori [20] suggests weaker conditons for "polynomial-time inferability". He allows the length of the longest input word so far read to be used as a parameter for the polynomial that bounds the number of implicit errors of prediction done.

4 Reversible and contextual languages

In this section we give definitions for reversible and contextual languages by using grammar formalism. Notice that these classes are usually defined by using automata notations.

Recall that a regular language is k-reversible, $k \geq 0$, if and only if whenever u_1vw and u_2vw are in L and $lg(v) = k$, then $T_L(u_1v) = T_L(u_2v)$ [3].

We give separate definitions for zero-reversible and k-reversible, $k \geq 1$, languages since it is possible to define the previous class by referring only to the productions of the grammar, while in the definition of the latter class we have to use derivations.

Definition 2 *A regular grammar $G = (V, \Sigma, P, S)$ is zero-reversible if the following conditions hold:*

(i) G is deterministic.
(ii) $A \to a$ and $B \to a$ imply $A = B$.
(iii) $A \to aC$ and $B \to aC$ imply $A = B$.

A regular language L is zero-reversible if there exists a zero-reversible regular grammar generating L.

The class of zero-reversible languages is denoted by $\mathcal{R}(0)$.

The three items of Definition 2 directly correspond to the following conditions guaranteeing a finite automaton A to be zero-reversible [3, p. 747]:

(i) A is deterministic,
(ii) A is reset-free,
(iii) A has at most one final state.

Recall that a finite automaton is reset-free if and only if for no two distinct states q_1 and q_2 do there exist an input symbol a and a state q such that $\delta(q_1, a) = \delta(q_2, a) = q$. Notice, that reset-freeness and one final state together indeed imply that two distinct nonterminals cannot have terminating productions with the same right hand side (cf. item (ii) of Definition 2).

If A is a finite automaton, then the *reverse automaton* A^r is obtained by reversing the direction of each transition in A. A finite automaton A is zero-reversible if and only if A and A^r both are deterministic [3]. Similar result can be easily proved in the grammar formalism. Instead of transitions, we now reverse productions.

Let $G = (V, \Sigma, P, S)$ be a regular grammar. The reverse grammar $H =$

(V, Σ, R, S) can be obtained as follows. If $S \to aA$ is in P, then take $A \to a$ to R. Similarly, if $A \to aB$ or $A \to a$ is in P, then take $B \to aA$ or $S \to aA$, respectively, to R. It is straightforward to verify that G is zero-reversible if and only if G and H both are deterministic.

The inference algorithm for $\mathcal{R}(0)$ is not quite as straightforward as the corresponding algorithm for $\mathcal{SZ}$. Namely, two appearances of a terminal b in sample words do not necessarily mean that they are produced by the same continuing production. Indeed, when applying the item (ii) of Definition 2 we need the additional information that the productions applied have exactly the same right hand sides before we can merge the corresponding left hand sides.

Example 3 *Consider a sample $\{abbbc, acc, abc\}$ from a zero-reversible language. We start with the following productions:*

$$A_1 \to aA_2, A_2 \to bA_3, A_3 \to bA_4, A_4 \to bA_5, A_5 \to c,$$

$$A_1 \to aA_6, A_6 \to cA_7, A_7 \to c,$$

and

$$A_1 \to aA_8, A_8 \to bA_9, A_9 \to c.$$

Notice how we apply here the earlier remark that the first character of a sample is produced by a production with the start symbol (A_1) in the left hand side. Contrary to the case of Szlard languages, we have to make the initial assumption "the same number of nonterminals as appearances of terminals".

By determinism, we have $A_2 = A_6 = A_8$, and by the item (ii) of Definition 4, we have $A_5 = A_7 = A_9$. This left us with the productions

$$A_1 \to aA_2, A_2 \to bA_3, A_3 \to bA_4, A_4 \to bA_5, A_5 \to c, A_2 \to cA_5, A_2 \to bA_5.$$

Now we can apply the item (iii) of Definition 4 to the productions $A_4 \to bA_5$ and $A_2 \to bA_5$ implying $A_4 = A_2$. Moreover, $A_2 \to bA_3$ and $A_2 \to bA_5$ imply $A_3 = A_5$, and further, $A_2 \to bA_3$ and $A_3 \to bA_3$ imply $A_2 = A_3$.

The inference process ends up with the productions

$$A_1 \to aA_2, A_2 \to bA_2, A_2 \to cA_2, A_2 \to c,$$

where A_1 is the start symbol.

It follows that the time complexity for outputting the next conjecture is not linear as in the case of $\mathcal{SZ}$ algorithm, but it has — at least in theory — a small non-linear factor [3]. We cannot use the off-line version of the set union algorithm because some of the merge operations to be done are made possible by earlier merges: we do not know all the merge operations to be done when the merging process begins.

Definition 4 *A regular grammar $G = (V, \Sigma, P, S)$ is k-reversible if it fulfils the following conditions:*

(i) G is deterministic.

(ii) the derivations

$$A_1 \Rightarrow a_1 A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1} A_k \Rightarrow a_1 \ldots a_k C$$

and

$$B_1 \Rightarrow a_1 B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1} B_k \Rightarrow a_1 \ldots a_k,$$

and the production $A_k \rightarrow b$ imply the existence of $B_k \rightarrow a_k C$.

(iii) the derivations

$$A_1 \Rightarrow a_1 A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_k A_{k+1}$$

and

$$B_1 \Rightarrow a_1 B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_k B_{k+1}$$

and the productions $A_{k+1} \rightarrow bA$ and $B_{k+1} \rightarrow bA$ imply $A_{k+1} = B_{k+1}$.

A regular language L is k-reversible if there exists a k-reversible regular grammar generating L.

The class of k-reversible languages is denoted by $\mathcal{R}(k)$. We have $\mathcal{R}(k) \subset \mathcal{R}(k+1)$ [3].

A finite automaton A is *deterministic with lookahead* k if, for any distinct pair of states q_1 and q_2, if q_1 and q_2 are both initial states or there is a state q such that $\{q_1, q_2\} \subseteq \delta(q, a)$, where a is a single input symbol of A, then there is no string w such that $lg(w) = k$ and both $\delta(q_1, w)$ and $\delta(q_2, w)$ are both defined (non-null).

The conditions (i)-(iii) of Definition 4 are justified by the fact that a finite automaton A is k-reversible if and only if it is deterministic and the reverse automaton A^r is deterministic with lookahead k [3, p. 749]. Muggleton [15, Fig. 6.9] has described this condition graphically.

Notice, that the initial states of A are final states in A^r, and that we have to apply the earlier remark concerning the situation where productions $A \rightarrow aB$ and $A \rightarrow a$ imply that B corresponds to a final state.

The item (ii) of Definition 4 corresponds to the case where, using the original automata terminology, two final states are merged if identical strings of length k lead to these states. However, merging two final states with no outgoing transitions causes no changes in the grammatical case, since there are no nonterminals correspondings to such final states. The item (ii) of Definition 4 handles the situation where at least one of the final states to be merged has an outgoing transition.

Example 5 *Consider a sample* $\{ab, bb, aab, abb\}$ *from a 1-reversible language. We start with the following productions:*

$$A_1 \to aA_2, A_2 \to b,$$

$$A_1 \to bA_3, A_3 \to b,$$

$$A_1 \to aA_4, A_4 \to aA_5, A_5 \to b,$$

and

$$A_1 \to aA_6, A_6 \to bA_7, A_7 \to b.$$

By determinism, we have $A_2 = A_4 = A_6$*. We next apply the item (ii) of Definiton 4. We have the derivations* $A_2 \Rightarrow bA_7$ *and* $A_3 \Rightarrow b$ *and the production* $A_2 \to b$*. These imply the existence of the production* $A_3 \to bA_7$*. Similarly, we have to add the productions* $A_5 \to bA_7$ *and* $A_7 \to bA_7$*.*

Now it is possible to apply the item (iii) of Definition 4. The derivations $A \Rightarrow aA_2$ *and* $A_2 \Rightarrow aA_5$ *and the productions* $A_2 \to bA_7$ *and* $A_5 \to bA_7$ *imply* $A_2 = A_5$*.*

In a similar manner we find the merge $A_3 = A_7$*. Since the nonterminals to be merged have the productions* $A_1 \to bA_3$*,* $A_3 \to bA_7$*, and* $A_3 \to b$*, the nonterminal* A_7 *corresponds to a final state. This means that we have to add the production* $A_1 \to b$ *to the grammar. This left us with the productions* $A_1 \to b$*,* $A_1 \to aA_2$*,* $A_1 \to bA_3$*,* $A_2 \to b$*,* $A_2 \to aA_2$*,* $A_2 \to bA_3$*,* $A_3 \to aA_3$*, and* $A_3 \to b$*, where* A_1 *is the start symbol. Since no further merges can be done, the inference process halts. This example is considered by Muggleton [15, pp. 101 and 108] using the automata formalism.*

Angluin [3] reports that the general algorithm for k-reversible languages runs in time $\mathcal{O}(kn^3)$, where n is the sum of the lengths of the input strings. Muggelton [15] has later shown that this algorithm can be implemented to run in time $\mathcal{O}(n^2)$.

Muggleton [15] argues that although reversible languages allow intuitively meaninful inferences processes, checking conditions of Definitions 2 and 4 is too slow for practical purposes. Moreover, no conjecture can be produced for a sample containing only one word. As a solution for these problems, he introduced the class on k-contextual languages [15].

Definition 6 *A regular grammar* $G = (V, \Sigma, P, S)$ *is* k*-contextual,* $k \geq 1$*, if it fulfils the following conditions:*

(i) G *is deterministic.*
(ii) the derivations

$$A_1 \Rightarrow a_1A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1}A_k \Rightarrow a_1 \ldots a_kC$$

and

$$B_1 \Rightarrow a_1 B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1} B_k \Rightarrow a_1 \ldots a_k,$$

and the production $A_k \to b$ *imply the existence of* $B_k \to a_k C$.

(iii) *the derivations*

$$A_1 \Rightarrow a_1 A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_k A_{k+1}$$

and

$$B_1 \Rightarrow a_1 B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_k B_{k+1}$$

imply $A_{k+1} = B_{k+1}$.

A regular language L *is* k*-contextual if there exists a* k*-contextual regular grammar generating* L.

The class of k-contextual languages is denoted by $\mathcal{C}(k)$.

Contrary to k-reversible languages, a pair of appearances of a substring of length k in the k-contextual case always causes a merge operation. Item (ii) of Definition 6 handles the case where one of the states to be merged is a final state with no outgoing transitions.

Now we can start the merging process with only one sample word. For example, if $ababd$ is a word in a 2-contextual language, we can start the inference process with this single input word. We have the derivation

$$S \Rightarrow aA_1 \Rightarrow abA_2 \Rightarrow abaA_3 \Rightarrow ababA_4 \Rightarrow ababd,$$

and according to Definition 6, we can merge A_2 and A_4. The inference process goes on as in the case of k-reversible languages, with the minor differences in the defining conditions given in Definitions 4 and 6. Hence, we simply test the grammar for conditions (i)-(iii), and merge nonterminals when the conditions are violated.

Clearly, we have $\mathcal{C}(k) \subset \mathcal{R}(k)$, $k \geq 1$.

In Definition 6 we have excluded the case $k = 0$, since 0-contextuality does not restrict the language in any meaninful way (cf. [15], Lemma 16.8). Moreover, 0-contextuality does have no natural interpretation in grammar formalism.

Ahonen [1] has generalized the idea of k-contextuality still further by considering classes of languages where the appearances of two subwords of length k not only imply that the subsequent elements are the same as in k-contextual languages, but that the subsequent elements are the same already after h, $h \leq k$, characters.

Definition 7 *Let* $1 \leq h \leq k$. *A regular grammar* $G = (V, \Sigma, P, S)$ *is* (k, h)*-contextual if it fulfils the following conditions:*

(i) *G is deterministic.*
(ii) *the derivations*

$$A_1 \Rightarrow a_1A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_h A_{h+1} \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1}A_k \Rightarrow a_1 \ldots a_k A_{k+1}$$

and

$$B_1 \Rightarrow a_1B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_h B_{h+1} \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1}B_k \Rightarrow a_1 \ldots a_k,$$

and the production $A_k \to a_k$ *imply* $A_i = B_i$, *for each* i, $i = h, \ldots, k$, *and the existence of the production* $B_k \to a_k A_{k+1}$.
(iii) *the derivations*

$$A_1 \Rightarrow a_1A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_h A_{h+1} \Rightarrow \ldots \Rightarrow a_1 \ldots a_k A_{k+1}$$

and

$$B_1 \Rightarrow a_1B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_h B_{h+1} \Rightarrow \ldots \Rightarrow a_1 \ldots a_k B_{k+1}$$

imply $A_{i+1} = B_{i+1}$, *for each* i, $i = h, \ldots, k$.

A regular language L *is* (k, h)*-contextual if there exists a* k, h*-contextual regular grammar generating* L.

The class of (k, h)-contextual languages is denoted by $\mathcal{C}(k, h)$.

When $h = k$, Definition 7 reduces to Definition 6. Hence, $\mathcal{C}(k, k) = \mathcal{C}(k)$. Similarly, it follows directly from the definitions that $\mathcal{C}(k, h) \subseteq \mathcal{R}(k)$ and, if $h < k$, $\mathcal{C}(k, h) \subseteq \mathcal{C}(k, h + 1)$.

The inference algorithm for (k, h)-contextual languages is as the k-contextual algorithm except that $k - h + 1$ merges instead of a single merge are done by items (ii) and (iii) of Definition 5.

Consider a regular language L over an alphabet Σ. Let I and F be subsets of Σ^{k-1}, and let T be a subset of Σ^k. Languages of the form $(I\Sigma^* \cap \Sigma^* F) \setminus \Sigma^* T \Sigma^*$ are referred to as *k-testable languages* [8]. The classes of k-contextual and k-testable languages coincide. The above definition of k-testable languages uses a set a forbidden substrings (T). As shown in [1], it is also possible to define this class of languages by using allowed substrings only.

Ahonen [1] gives efficient linear time inference algorithms based on the substring representation both for k-contextual and (k, h)-contextual languages.

5 Strictly regular and code regular languages

The class of k-contextual languages is obtained by tightening the definition of k-reversibility. Another way for defining new subclasses of regular languages inferable from positive data is to relax the definition of $\mathcal{SZ}$. A regular grammar generating a language in $\mathcal{SZ}$ has unique terminals in the right hand sides of its productions. If we replace unique terminals by unique strings with the additional property that each such string begins with a different character, we come to the concept of strictly deterministic regular languages [20]. If the set of the unique strings is a code (for codes and languages, see e.g. [17]) we have a code regular language [6].

Yokomori [20] defined strictly deterministic languages by extending the notation of finite automata such that arbitrary non-empty strings are allowed in transitions. We could follow this line and allow arbitrary non-empty strings in the productions: a continuing production would have the form $A \to wB$ and a terminating production would have the form $A \to w$, where $w \in \Sigma^+$. However, we like to obey the earlier restrictions concerning the form of productions in regular grammars. This leads us to the following definitions.

Let $G = (V, \Sigma, P, S)$ be a regular grammar. A nonterminal A is *single* in G if there is only one A-production in P. Otherwise, A is *branching*. The start symbol S is always considered to be branching, irrespective of the number of S-productions.

Definition 8 *A regular grammar $G = (V, \Sigma, P, S)$ is strictly deteministic if, for all branching nonterminals A and B, productions $A \to a\alpha$ and $B \to a\beta$, $\alpha, \beta \in (\Sigma \cup \{\lambda\})$, imply $A = B$.*

A regular language is strictly deterministic if there exists a strictly deterministic regular grammar generating L.

The class of strictly deterministic regular languages is denoted by $\mathcal{S}$. See [18] for a slightly different definition of strictly deterministic languages.

The uniqueness requirement of regular grammmars generating languages in $\mathcal{SZ}$ does not hold here for productions $A \to a\alpha$, where A is single and $\alpha \in ((V \setminus \Sigma) \cup \{\lambda\})$. Such a may well appear in the right hand side of several other productions with a single nonterminal in the left hand side and in at most one production with a branching nonterminal in the left hand side.

Given a sample, the problem is to find the terminals produced by productions with a branching production in the left hand side. This is a kind of parsing problem which is possible to solve because of the uniqueness of the terminals appearing in the corresponding right hand sides.

In order to "parse" the sample, we have to find the string segments of the form $w = a_1 \ldots a_k$ defined by the derivations

$$A \Rightarrow a_1 A_1 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{k-1} A_{k-1} \Rightarrow a_1 \ldots a_k B,$$

where A is branching, A_i, $i = 1, \ldots, k-1$ are single, and B is branching or $B = \lambda$. The set of such strings $w = a_1 \ldots a_k$ in G is denoted by M_G.

The inference algorithm for $\mathcal{S}$ essentially maintains the set of segments parsing the current sample. When the segments are found, the inference algorithm reduces to that used for $\mathcal{SZ}$. Namely, if L is in $\mathcal{S}$, there is a language K in $\mathcal{SZ}$ such that $h(K) = L$, where h is a bijective homomorphism. If aw, $w \in \Sigma^*$, is in M_G, then we can set $h(a) = aw$.

Yokomori [20] has shown that it is possible to implement an inference algorithm for $\mathcal{S}$ running on time $\mathcal{O}(\mid \Sigma \mid m)$, where $\mid \Sigma \mid$ is the size of the alphabet used and m is the length of the longest input word. He has also shown that the same bound hold for the implicit errors of prediction made. Hence, strictly deterministic languages are polynomial-time inferable in Yokomori's sense [20].

Example 9 *Consider a sample $\{bcc, bcdabc, bcdcaa\}$ from a strictly deterministic language. The longest common prefix bc of the sample word is one of the string segment. There can be no other segments beginning with b. The word bcc must consists of two segments; the only segment beginning with c is c itself. The longest common prefix of the subwords $dabc$ and $dcaa$ is d, which is the segment beginning with d. Since c is already known to a segment, we determine the longest common prefix of abc and aa, and find out the the segment beginning with a is a itself.*

The idea of strictly deterministic languages is generalized further by Emerald et al. [6] as follows.

Definition 10 *A regular grammar $G = (V, \Sigma, P, S)$ is code regular if the set M_G is a code.*

The class of code regular languages is denoted by $\mathcal{D}$. By definition, we have $\mathcal{S} \subset \mathcal{D}$.

An efficient inference algorithm is known to exist only when the code over M_G is both a prefix code (no word in M_G is a proper prefix of another word in M_G) and a suffix code (no word in M_G is a proper suffix of another word in M_G) [6]. The difference between the inference algorithms for $\mathcal{S}$ and $\mathcal{D}$ is in the method for finding the current segments.

6 Uniquely terminating regular languages

The last class of languages considered in this paper is called the class of uniquely terminating regular languages. Notice carefully the difference between item (ii) of Definition 2 and item (i) of the following definition.

Definition 11 *A regular grammar $G = (V, S, P, S)$ is uniquely terminating if the productions in P fulfil the following conditions for each nonterminal A in G:*

(i) $A \to aB$ and $A \to aC$ imply $B = C$.
(ii) A has a unique terminating production; i.e. each nonterminal has exactly one terminating production. The terminals appearing in the right hand sides of terminating productions are all different.

A regular language L is said to be uniquely terminating if there is a uniquely terminating grammar generating L.

The class of uniquely terminating regular languages is by $\mathcal{U}$.

Notice that the right hand side b of a unique terminating production $A \to b$ may appear in the right hand side of any continuing production, but not in the right hand side of any other terminating production.

Suppose we are given a set of sample words $\{w_1, w_2, ..., w_n\}$ from a uniquely terminating regular language. We can give the corresponding derivations in the form

$$S \Rightarrow w_{i1}A_{i1} \Rightarrow \ldots \Rightarrow w_{i1} \ldots w_{ik}A_{ik} \Rightarrow w_{i1} \ldots w_{ik}w_{ik+1},$$

where $w_i = w_{i1} \ldots w_{ik}w_{ik+1}$, for $i = 1, ..., n$, with the appropriate length $k + 1$. We can take the following steps when merging nonterminals A_{ij} in the inference algorithm:

(i) If two sample words have a common proper prefix w, we know by the item (i) of Definition 11 that the common proper prefix is produced by using the same productions in both derivations. This means that we can merge the corresponding nonterminals.
(ii) By the uniqueness of terminating productions (the item (ii) of Definition 11), we can merge the last nonterminals of derivations producing words which end with the same terminal. For example, if the sample contains words b and $cdbab$, we know that the last nonterminal appearing in the derivation of the longer sample word is the start symbol of the grammar in question.

Since terminating productions are unique, $\mathcal{SZ}$ is not included in $\mathcal{U}$. For example, the language $\{ab, ac\}$ cannot be in $\mathcal{U}$. On the other hand, $\mathcal{U}$ contains languages which are not in $\mathcal{R}(k)$, for any value of k [14].

7 Conclusions

We have surveyed several subclasses of regular languages inferable from positive data. We defined the classes in a uniform grammatical manner. We hope that this uniform way of defining the classes help the further research to find new subclasses of regular languages inferable from positive data and new relationships between the known classes of languages.

References

[1] H. Ahonen, Generating grammars for structured documents using grammatical inference methods, Ph. D. Thesis, Department of Computer Science, University of Helsinki, Report A-1996-4, 1996.

[2] D. Angluin, Finding patterns common to a string, *J. Comput. Syst. Sci.* **21** (1980), 46–62.

[3] D. Angluin, Inference of reversible languages, *J. ACM* **29** (1982), 741–765.

[4] D. Angluin and C.H. Smith, Inductive inference: theory and methods, *ACM Comput. Surv.* **15** (1983), 237–269.

[5] S. Crespi-Reghizzi, G. Guida, and D. Mandrioli, Noncounting context-free languages, *J. ACM* **25** (1978), 571-580.

[6] J.D. Emerald, K.G. Subramanian and D.G. Thomas, Learning code regular and code linear languages, in: *Proceedings of ICGI-96, Lecture Notes in Artificial Intelligence* **1147** (1996), 211–221.

[7] H.N. Gabow and R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, in: *Proc. 15th ACM Symposium on Theory of Computing*, (1983), 246–251.

[8] P. Garcia, E. Vidal and J. Oncina, Learning locally testable languages in the strict sense, in: *Proceedings of the First International Workshop on Algorithmic Learning Theory* (1990), 325–338.

[9] E.M. Gold, Language identification in the limit, *Inform. Contr.* **10** (1967), 447–474.

[10] M.A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.

[11] E. Mäkinen, On context-free derivations, *Acta Univ. Tamper. Ser. A* **198**, (1985).

[12] E. Mäkinen, The grammatical inference problem for the Szilard languages of linear grammars, *Inf. Process. Lett.* **36** (1990), 203–206.

[13] E. Mäkinen, A family of languages which is polynomial-time learnable from positive data in Pitt's sense, *Int. J. Computer Math.* **61** (1996), 175–179.

[14] E. Mäkinen, Inferring uniquely terminating regular languages from positive data, To appear in *Inf. Process. Lett.*

[15] S. Muggleton, *Inductive Acquisition of Expert Knowledge*, Addison-Wesley, 1990.

[16] L. Pitt, Inductive inference, DFAs, and computational complexity in: *Proceedings of 2nd Workshop on Analogical and Inductive Inference, Lecture Notes in Artificial Intelligence* **397** (1989), 18–44.

[17] A. Salomaa, *Jewels of Formal Language Theory*, Computer Science Press, 1981.

[18] N. Tanida and T. Yokomori, Polynomial-time identification of strictly regular languages in the limit, *IEICE Trans. Inf. & Syst.* **E75-D** (1992), 125–132.

[19] R.E. Tarjan and J. van Leeuwen, Worst-case analysis of set union algorithms, *J. ACM* **31** (1984), 245–281.

[20] T. Yokomori, On polynomial-time learnability in the limit of strictly regular languages, *Machine Learning* **19** (1995), 153–179.