Permutations and Control Sets for Learning Non-regular Language Families

Henning Fernau[1] and José M. Sempere[2]

[1] Wihelm-Schickard-Institut für Informatik, Universität Tübingen, D-72076 Tübingen, Germany, Email: fernau@informatik.uni-tuebingen.de
[2] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Valencia 46071, Spain, Email: jsempere@dsic.upv.es

Abstract. We discuss two versatile methods which can be used to transfer learnability results from one language class to another. We apply these methodologies to three learning paradigms: (1) Learning in the limit, (2) Morphic generator grammar inference, and (3) Query learning.
Keywords:Formal languages, universal grammars, control sets, learning from positive data.

1 Introduction

In this paper, we will present two methods for transferring learnability results from one language class to another by simple preprocessing. We mainly concentrate on the paradigm "learning in the limit from positive data". It is not hard to see that similar techniques can be used to develop efficient learning algorithms in other paradigms as well. In the case of query learning, this has been done (within the framework of matrix grammars, see below) in [9]. We will detail such extensions at the end.

Here, we will focus on the following version of the learning model "identification in the limit" proposed by Gold [18]:

- An inference machine (a "learner") IM is given the task to infer a language from a certain fixed language class $\mathcal{F}$ for which a description formalism (in our case, a grammar formalism) is also fixed.
- To the inference machine IM, a language $L \in \mathcal{F}$ is presented by giving all the elements of L to IM one by one (maybe, with repetitions), i.e., $L = \{ w_i \mid i \geq 0 \}$, and w_i is given to IM at time step i.
- After having received w_i, IM responds with a hypothesis grammar G_i. Of course, we can see G_i as the result of computing a recursive $(i+1)$-ary function f_i:

$$G_i = f_i(w_0, \ldots, w_i). \tag{1}$$

A.L. Oliveira (Ed.): ICGI 2000, LNAI 1891, pp. 75–88, 2000.

The inference machine IM is called a *learner for* $\mathcal{F}$ if

1. the process described above always converges in the discrete space of $\mathcal{F}$-grammars, i.e., for all presentations $\{w_i \mid i \geq 0\} \in \mathcal{F}$, the corresponding grammar sequence $G_0, G_1, G_2, \ldots$ converges to a limit grammar G, which means that there is an i_0 such that for all $i \geq i_0$ we find $G = G_{i_0} = G_i$;
2. the limit grammar G is independent of the presentation of the language L and, moreover, $L \subseteq L(G)$.

Note that there are a lot of language families known to be identifiable in the limit from positive data. According to Gregor [19], the most prominent examples of identifiable regular language families are:

- k-testable languages [15,17] (see below),
- k-reversible languages [4] and
- terminal distinguishable regular languages [30,31].

Generalizations of these language classes are discussed in [1,10,11,21,32]. Further identifiable language families, especially also non-regular ones, can be found as references in the quoted papers.

All these language classes can be learned *efficiently*, i.e., the time complexity for the computation of the hypothesis function(s) f_i in Eq. (1) is only polynomial in the size of its input, which is the total length of the input sample words up to step i.

2 Formal Language Definitions

Notations: Σ^k is the set of all words of length k over the alphabet Σ, $\Sigma^{<k} = \bigcup_{j=0}^{k-1} \Sigma^j$, and $\Sigma^* = \bigcup_{j \geq 0} \Sigma^j$. Moreover, $|w|$ denotes the length of $w \in \Sigma^*$, i.e., $|w| = k$ iff $w \in \Sigma^k$. λ denotes the empty word, which is the only word of length 0.

Definition 1. *A* linear grammar *is a construct $G = (N, \Sigma, P, S)$, where N is the finite set of nonterminals, $S \in N$ is the start symbol, and P is a set of linear rules, which are rules of the form $A \to vBw$ and $A \to w$, where $A, B \in N$ and $v, w \in \Sigma^*$. G defines a derivation relation $\Rightarrow$ via $x \Rightarrow y$ iff $\exists \alpha, \beta \in \Sigma^*, A \to u \in P : x = \alpha A \beta$ and $y = \alpha u \beta$. G* generates *the language $L(G) = \{S \stackrel{*}{\Rightarrow} w \mid w \in \Sigma^*\}$, where $\stackrel{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$. A language is linear if there exists a linear grammar generating this language.*

Of special interest for our purpose are the so-called even linear languages *(introduced by Amar and Putzolu in [2]) characterized by even linear grammars, where the rules of the form $A \to vBw$ obey $|v| = |w|$.*

Furthermore, the regular languages *are characterized via* right-linear grammars*, which are linear grammars where the rules of the form $A \to vBw$ obey $w = \lambda$.*

Let REG denote the family of regular languages, ELL the family of even linear languages and LIN the family of linear languages. REG $\subsetneq$ ELL $\subsetneq$ LIN is well-known [2,33].

In order to provide a simple running example, we define:

Definition 2. *A language L is* k-testable (in the strict sense)*, or k-TLSS for short, iff $L = I_k\Sigma^* \cap \Sigma^* F_k \setminus \Sigma^* T_k \Sigma^*$, where $I_k, F_k \subseteq \Sigma^{<k}$ and $T_k \subseteq \Sigma^k$.*

3 Permutation Families for Learning from Positive Data

In order to present our results, we need some further notions.
Let $\psi_n : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be a *permutation*, i.e., a bijective mapping. A collection $\Psi = (\psi_n \mid n \geq 1)$ of permutations is called a *family of permutations*. Ψ is called *uniformly polynomial-time computable* if there is an algorithm $\mathbb{A}_f$ realizing the partial function $f : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ with $f(n, m) = \psi_n(m)$ in polynomial time and another algorithm $\mathbb{A}_g$ realizing the partial function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with $g(n, m) = \psi_n^{-1}(m)$ in polynomial time. To every family of permutations $\Psi = (\psi_n \mid n \geq 1)$, there corresponds a *family of inverse permutations* $\Psi^{-1} = (\psi_n^{-1} \mid n \geq 1)$.

Example 3. Consider

$$\psi_n(m) = \begin{cases} 2m - 1 & , \text{if } m \leq (n+1)/2 \\ 2(n - m + 1), & \text{if } m > (n+1)/2 \end{cases}$$

Both $\psi_n(m)$ and its inverse are easily seen to be polynomial-time computable. Hence $\Psi = (\psi_n \mid n \geq 1)$ is a polynomial-time computable family of permutations.

It is easy to construct further uniformly polynomial-time computable families of permutations from known ones. This can be done, e.g., by the following two operations (here, let $\Psi = (\psi_n)$ and $\Phi = (\phi_n)$ be uniformly polynomial-time computable families of permutations):

piecewise mixture Let n_0 be fixed. Define $\Xi = (\xi_n)$ by $\xi_n = \psi_n$, if $n < n_0$, and $\xi_n = \phi_n$ otherwise.

permutationwise composition Define $\Xi = (\xi_n)$ by $\xi_n = \psi_n \circ \phi_n$ with $\xi_n(x) = \psi_n(\phi_n(x))$.

For example, the permutationwise composition of Ψ as defined in Example 3 with itself yields the family $\Xi = (\xi_n)$, where, e.g., ξ_{11} can be described by the following table:

$x =$	1	2	3	4	5	6	7	8	9	10	11
$\psi_{11}(x) =$	1	3	5	7	9	11	10	8	6	4	2
$\xi_{11}(x) =$	1	5	9	10	6	2	4	8	11	7	3

Now, let us fix an alphabet Σ and a family of permutations $\Psi = (\psi_n \mid n \geq 1)$. In order to avoid further awkward notations, let us denote by $\Psi(w)$, where $w = a_1 \ldots a_n \in \Sigma^n$, the word $\Psi(w) = a_{\psi_n(1)} \ldots a_{\psi_n(n)}$. We extend this notation further to language by setting, for $L \subseteq \Sigma^*$, $\Psi(L) = \{\Psi(w) \mid w \in L\}$ and to language families (obviously not necessarily restricted to a specific alphabet Σ anymore) by defining $\Psi(\mathcal{L}) = \{\Psi(L) \mid L \in \mathcal{L}\}$.

The following theorem is easily shown:

Theorem 4. *If $\mathcal{L}$ is a language family for which a polynomial-time learning algorithm in the limit from positive data is known, then $\Psi(\mathcal{L})$ can be learned in polynomial-time in the limit from positive data as well, if Ψ is a uniformly polynomial-time computable family of permutations.*

Proof. Let $\mathbb{A}_{\mathcal{L}}$ be the learning algorithm for language family $\mathcal{L}$. The learning algorithm for $\Psi(\mathcal{L})$ uses $\mathbb{A}_{\mathcal{L}}$ as a subroutine in the sense that it translates its input sequence $w_1, w_2, w_3, \ldots$ into an input sequence $\Psi^{-1}(w_1), \Psi^{-1}(w_2), \Psi^{-1}(w_3), \ldots$ of algorithm $\mathbb{A}_{\mathcal{L}}$. To this input sequence, $\mathbb{A}_{\mathcal{L}}$ responds with a sequence of grammars $G_1, G_2, G_3, \ldots$ generating the languages $L_1, L_2, L_3, \ldots$. Now, the intended learning algorithm just interprets $G_1, G_2, G_3, \ldots$ as representing languages

$$\Psi(L_1), \Psi(L_2), \Psi(L_3), \ldots \ .$$

The correctness and efficiency of the described algorithm trivially carries over from $\mathbb{A}_{\mathcal{L}}$, since Ψ was assumed to be a uniformly polynomial-time computable family of permutations. □

Of course, it is a bit abstract and unnatural to consider a grammar G of L to represent "suddenly" a language $\Psi^{-1}(L)$ as done in the preceding proof. We will show natural examples of such an interpretation in the following by making use of the concept of control sets.

4 Control Sets for Learning from Positive Data

Let G denote some grammar.[1] A word w (over G's terminal alphabet) belongs to the language $L(G)$ generated by G iff there is a sequence of rules $r_1 \ldots r_m$ whose sequential application, starting from an axiom of G, yields w. In other words, w is somehow described by a word (called *control word* or *associate word*) over the rule set P of G. If G is ambiguous, this description is not unique. On the other hand, one could consider the sublanguage $L(G, R)$ of $L(G)$ consisting of those (terminal) words derivable through G which have control words in $R \subseteq P^*$. Here, R is called *control set*.[2] If $\mathcal{G}$ is some grammar family and $\mathcal{L}$ is some language family, then let

$$\mathrm{CONTROL}(\mathcal{G}, \mathcal{L}) = \{L(G, R) \mid G \in \mathcal{G}, L \in \mathcal{L}\}\,.$$

Let us fix, for the moment, some grammar family $\mathcal{G}$. A grammar $G_0^{\Sigma} \in \mathcal{G}$ is called *base grammar* for the alphabet Σ if $L(G_0^{\Sigma}) = \Sigma^*$ and, furthermore, every word in Σ^* can be derived unambiguously via G_0^{Σ}. Let $\mathcal{G}_0 \subset \mathcal{G}$ be a collection of base grammars such that for every (with respect to $\mathcal{G}$) possible terminal alphabet Σ, there is exactly one base grammar in $\mathcal{G}_0$. Finally, grammar

[1] Similar notions can be developed for automata.

[2] For many properties of controlled language families, we refer to [20].

subfamily $\mathcal{G}_0$ is called *universal* if $\mathrm{CONTROL}(\mathcal{G}_0, \mathrm{REG}) = L(\mathcal{G})$, where $L(\mathcal{G})$ denotes the language family generated by the grammar family $\mathcal{G}$.

In general, there are various universal grammar subfamilies. Consider, for example, the following:

Example 5. Both $\{G_0^\Sigma \mid \Sigma \text{ is an alphabet}\}$, where

$$G_0^\Sigma = (\{S\}, \Sigma, \{S \to \lambda\} \cup \{S \to aS \mid a \in \Sigma\}, S)$$

and, more generally, all families $\mathcal{G}_{0,k} = \{G_{0,k}^\Sigma \mid \Sigma \text{ is an alphabet}\}$, where

$$G_{0,k}^\Sigma = (\{S\}, \Sigma, \{S \to x \mid x \in \Sigma^{<k}\} \cup \{S \to uS \mid u \in \Sigma^k\}, S)\,,$$

are universal for REG, i.e., $\mathrm{CONTROL}(\mathcal{G}_0, \mathrm{REG}) = \mathrm{REG}$.

As within the permutation approach, we can use use universal grammar families in order to obtain (new) learnable language families through learnable control set classes. More specifically, we can present the following learning algorithm for $\mathrm{CONTROL}(\mathcal{G}_0, \mathcal{L})$, given any learnable language class $\mathcal{L}$:

1. Consider a new input word w_j.
2. w_j is transformed to the unique control word π_j.
3. π_j is given to the identification algorithm $\mathbb{A}_\mathcal{L}$ of $\mathcal{L}$.
4. $\mathbb{A}_\mathcal{L}$ outputs hypothesis grammar G_j.
5. The whole algorithm outputs hypothesis language $L(G_0^\Sigma, L(G_j))$, where Σ is the alphabet of symbols contained in $w_1 \ldots w_j$.

The algorithm is efficient if its second step can be performed in polynomial time. We will assume this to be the case in the following. In a certain sense, grammar G_j can be viewed as "representing" $L(G_0^\Sigma, L(G_j))$. More precisely, since $\mathcal{G}_0$ is universal, for every G_j there exists a $H_j \in \mathcal{G}$ such that $L(H_j) = L(G_0^\Sigma, L(G_j))$. If the transformation $G_j \mapsto H_j$ can be done efficiently[3], the above algorithm could indeed give a hypothesis grammar, namely H_j, in its last step.

Remark 6. As described above, each grammar $G \in \mathcal{G}_0$ translates a string over, say, Σ into another string over another alphabet. For example, each $G_{0,k}^\Sigma$ (as defined in Example 5) can be viewed as a deterministic generalized sequential machine.[4] This view immediately explains two things:

- The transformation of a word w into its control word (with respect to $G_{0,k}^\Sigma$) can be done in linear time.
- If $\mathcal{L}$ is a trio, then $\mathrm{CONTROL}(\mathcal{G}_{0,k}, \mathcal{L}) = \mathcal{L}$ for each $k \geq 1$ due to the Theorem of Nivat [6,27].

Actually, $G_{0,k}^\Sigma$ realizes some sort of tape compression. We will consider words over $\Delta = \Sigma^{\leq k}$ as control words for $G_{0,k}^\Sigma$.

[3] This is indeed the case for all published learning applications of control languages we know of.

[4] For notions like generalized sequential machines, trios, etc. we refer to [6,22,27].

Note that even considering identifiable subclasses of the regular languages may yield new interesting identifiable subclasses in this way.

Theorem 7. *We continue Example 5. For all* $k, l \in \mathbb{N}$, $CONTROL(\mathcal{G}_{0,k}, \ell\text{-}TLSS)$ *is efficiently identifiable from positive samples.*[5] *Moreover,*

1. $CONTROL(\mathcal{G}_{0,1}, \ell\text{-}TLSS) = \ell\text{-}TLSS \subsetneq CONTROL(\mathcal{G}_{0,\ell}, 2\text{-}TLSS)$,
2. $CONTROL(\mathcal{G}_{0,k}, \ell\text{-}TLSS) \subsetneq CONTROL(\mathcal{G}_{0,k}, \ell+1\text{-}TLSS)$,
3. $CONTROL(\mathcal{G}_{0,k}, \ell\text{-}TLSS) \subsetneq CONTROL(\mathcal{G}_{0,k+1}, \ell\text{-}TLSS)$,
4. $CONTROL(\mathcal{G}_{0,k}, \ell\text{-}TLSS) \subsetneq CONTROL(\mathcal{G}_{0,k\ell}, 2\text{-}TLSS)$.

Proof. We have only to show the "moreover-part":

1. CONTROL$(\mathcal{G}_{0,1}, \ell$-TLSS$) = \ell$-TLSS is clear by definition. In order to show ℓ-TLSS $\subseteq$ CONTROL$(\mathcal{G}_{0,\ell}, 2$-TLSS$)$, recall that languages from ℓ-TLSS can "test" prefixes and suffixes of length $\ell - 1$ and forbidden subwords of length ℓ. On the other hand, the input word $w \in \Sigma^*$ of the combined learning algorithm will be essentially sliced into parts $w = u_1 \dots u_n$, where $u_i \in \Sigma^\ell$ and $u_n \in \Sigma^{<\ell}$, if we let words v from Σ^ℓ denote the rule $S \to vS$ in $G^{\Sigma}_{0,\ell}$ and words $v \in \Sigma^{<\ell}$ denote the rule $S \to v$ in $G^{\Sigma}_{0,\ell}$. Note that $u_1, \dots, u_n$ are the input symbols of the word given to the 2-TLSS algorithm.
   Consider now $L = I_\ell \Sigma^* \cap \Sigma^* F_\ell \setminus \Sigma^* T_\ell \Sigma^*$, where $I_\ell, F_\ell \subseteq \Sigma^{<\ell}$ and $T_\ell \subseteq \Sigma^\ell$. Let $I'_\ell = (I_\ell \Sigma^* \cap \Sigma^{\ell-1}) \cup (I_\ell \Sigma^* \cap \Sigma^* F_\ell \cap \Sigma^{<\ell-1})$ and $F'_\ell = (\Sigma^* F_\ell \cap \Sigma^{\ell-1}) \cup (I_\ell \Sigma^* \cap \Sigma^* F_\ell \cap \Sigma^{<\ell-1})$. Observe that $L = I'_\ell \Sigma^* \cap \Sigma^* F'_\ell \setminus \Sigma^* T_\ell \Sigma^*$ and $I'_\ell \cap F'_\ell \cap \Sigma^{<\ell-1} = L \cap \Sigma^{<\ell-1}$.
   We have to design a language $\hat{L} = \hat{I}_2 \Delta^* \cap \Delta^* \hat{F}_2 \setminus \Delta^* \hat{T}_2 \Delta^* \subset \Delta^*$, where $\Delta = \Sigma^{\leq \ell}$.
   Let us first consider "short words" u of length $< \ell$ in L. Hence, u has to be in the control set, which is guaranteed when u is both in $\hat{I}_2$ and in $\hat{F}_2$. In particular, $\hat{I}_2 \cap \Sigma^{<\ell} = L \cap \Sigma^{<\ell}$.
   In the following, we restrict our discussions to words of length at least ℓ. Now, p is a prefix of length $\ell - 1$ of w iff p is prefix of u_1, a property which can be tested easily by the control language which is from 2-TLSS. More specifically, $\hat{I}_2 \cap \Sigma^\ell = \{pa \mid p \in I'_\ell, a \in \Sigma\}$.
   The set of subwords of length ℓ of w equals the set of subwords of length ℓ of the language $\{u_i u_{i+1} \mid 1 \leq i < n\}$, so that forbidden subwords of length ℓ of w can be tested through forbidden subwords of length 2 of the control language.
   Finally, let s be a suffix of length $\ell - 1$ of w. This basically means that we have to "allow" all suffixes uv of control words, where $u \in \Sigma^\ell$, $v \in \Sigma^{<\ell}$ and s is suffix of uv. In particular, this means that s is in $\hat{F}_2$. But this is not enough. We have to put all suffixes of s in $\hat{F}_2$ and forbid $T = \{uv \mid u \in \Sigma^\ell, v \in \Sigma^{<\ell}, \forall s \in F'_\ell \cap \Sigma^{\ell-1} : s \neq v \text{ and } s \text{ is not suffix of } uv\}$.
   The inclusion is strict since $\Sigma^{2\ell-1} \in$ CONTROL$(\mathcal{G}_{0,\ell}, 2$-TLSS$) \setminus \ell$-TLSS.

[5] [17] contains an algorithmic definition of an automata family characterizing k-TLSS.

2. & 3. The inclusions themselves are trivial. $L = \{w \in \Sigma^* \mid |w| \leq k\ell + 1\} \notin \text{CONTROL}(\mathcal{G}_{0,k}, \ell\text{-TLSS})$, but L lies in both $\text{CONTROL}(\mathcal{G}_{0,k}, \ell+1\text{-TLSS})$ and $\text{CONTROL}(\mathcal{G}_{0,k+1}, \ell\text{-TLSS})$.
4. This is a straightforward generalization of the first item. □

Example 8. $\{aa\} \in$ 3-TLSS. This yields the control "word" $[aa]$ of length 1 (codifying the rule $S \to aa$) via $G_{0,3}^{\{a\}}$. Obviously, $[aa] \in$ 2-TLSS.

5 Putting Things Together

Let $\Psi = (\psi_n)$ and $\Phi = (\phi_n)$ be uniformly polynomial-time computable families of permutations. Let $\mathcal{G}_0$ be a universal subfamily of the grammar family $\mathcal{G}$. If language family $\mathcal{L}(\mathcal{G})$ is efficiently identifiable from positive samples only, then the following algorithm is also efficient:

1. Permute an input word w_j according to Ψ^{-1}, yielding $w'_j = \Psi^{-1}(w_j)$.
2. Compute the control word π_j of w'_j according to a suitable $G_0 \in \mathcal{G}_0$.
3. Permute π_j according to Φ^{-1}, yielding $\pi'_j = \Phi^{-1}(\pi_j)$.
4. The identification algorithm, given π'_j, yields a grammar G_j.
5. The new guess of the whole algorithm is $\Psi(L(G_0, \Phi(L(G_j))))$.

Here, we get the problem of which language family $\mathcal{L}(\Psi, \mathcal{G}_0, \Phi, \mathcal{L}(\mathcal{G}))$ will be identified using such a mixed strategy. In some special cases, we could give characterizations of those language families, and we will focus on those families in the following. To this end, let $\Phi = \mathbb{ID}$ be the family of identities.

Example 9. Let Ψ be defined as in Example 3 and $\mathcal{G}_{0,2}$ as in Example 5. Then, $\mathcal{L}(\Psi, \mathcal{G}_{0,2}, \mathbb{ID}, \text{REG}) = \text{ELL}$. This can be easily seen by observing the following facts:

1. The grammars $H_0^\Sigma = (\{S\}, \Sigma, \{S \to aSb \mid a, b \in \Sigma\} \cup \{S \to x \mid x \in \Sigma^{<2}\}, S)$ are universal for the even linear languages, see [33,23,24].
2. Control words for H_0^Σ can be viewed as words over Δ, where $\Delta = \Sigma^{\leq 2}$. Observe that words over Δ can be viewed as control words for $G_{0,2}^\Sigma$ as well.
3. If $w \in \Sigma^*$ has the control word π according to H_0^Σ, then $\Psi(w)$ has the control word π according to $G_{0,2}^\Sigma$ and vice versa.

Hence, in particular,

$$\mathcal{L}(\Psi, \mathcal{G}_{0,2}, \mathbb{ID}, \text{2-TLSS}) = \text{CONTROL}(\{H_0^\Sigma \mid \Sigma \text{ is an alphabet}\}, \text{2-TLSS})$$

is identifiable from positive samples only.

The following observation is an immediate corrollary of the definitions. It is interesting, since semilinear properties generally need non-trivial proofs.[6]

[6] The notion of semilinearity is explained, e.g., in [22]. It is important both from a linguistic point of view and from the standpoint of learning algorithms, cf. [8,35].

Corollary 10. *Fix $k \geq 1$. Let Ψ and Φ be arbitrary families of permutations. Then, $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, REG)$ contains only semilinear languages, where $\mathcal{G}_{0,k}$ is defined as in Example 5.*

In the following, we restrict the notion of universal grammar family further:

Definition 11. *Let us call a subfamily $\mathcal{G}_0 = \{G_0^\Sigma \mid \Sigma \text{ is an alphabet}\}$ of linear grammars* uniformly described *if every universal grammar G_0^Σ contains exactly one nonterminal S and the rules of G_0^Σ can be characterized by a pair of natural numbers (n, m) such that $S \to \alpha$ with $\alpha \in \Sigma^*$ is a rule iff $|\alpha| < n + m$ and $S \to \alpha S \beta$ is a rule iff $\alpha \in \Sigma^n$ and $\beta \in \Sigma^m$.*

One can easily prove:

Lemma 12. *If $\mathcal{G}_0$ is a uniformly described grammar family, then $CONTROL(\mathcal{G}_0, REG) \subseteq LIN$.*

In the proof of the preceding lemma, the regular control language given by, e.g., a deterministic finite automaton, is simulated in the nonterminals which basically store the state of the automaton.

For example, the universal grammar subfamilies presented for REG in Example 5 are uniformly described, as well as the universal grammar subfamily for ELL in Example 9.

Theorem 13. *If $\mathcal{G}_0$ is a uniformly described universal family of linear grammars, then there exists a uniformly polynomial-time computable family of permutations Ψ such that $\Psi(L(\mathcal{G}_0)) = REG$.*

Proof. (Sketch) Let $G_0^\Sigma \in \mathcal{G}$ be described by the number pair (n, m). Let $\Psi = (\psi_\ell \mid \ell \in \mathbb{N})$ be recursively defined as:

$$\begin{array}{lll} \psi_\ell(\nu) & = \nu, & \text{if } \ell < n + m \vee \nu \leq n \\ \psi_\ell(\ell - \nu + 1) & = n + \nu, & \text{if } \ell \geq n + m \wedge \nu \leq m \\ \psi_\ell(\nu) & = n + m + \psi_{\ell - n - m}(\nu - n), & \text{if } \nu \in (n, \ell - m) \end{array}$$

□

Example 14. Considering the rule pattern $S \to uSv$, $S \to u$ and $S \to \lambda$ where u, v are terminal placeholders (this corresponds to Example 9), we obtain a grammar family which characterizes ELL. The corresponding family of permutations can be defined recursively according to the preceding proof as:

$$\psi_n(m) = \begin{cases} 1 & , \text{ if } m = 1 \\ 2 & , \text{ if } m = n \\ \psi_{n-2}(m-1) + 2, & \text{if } m \in (1, n) \end{cases}$$

By an obvious induction argument, one sees that this family of permutations is the same as that in Example 3.

As a simple corollary of our previous discussions, we can state:

Corollary 15. *Consider a language family $\mathcal{L} \subset REG$ for which a polynomial-time learning algorithm in the limit from positive data is known and a uniformly described universal grammar family $\mathcal{G}_0$. Let Ψ be the family of permutations defined in the previous theorem. Then $\Psi^{-1}(\mathcal{L}) \subset LIN$ can be learned in polynomial-time in the limit from positive data as well. Moreover, there is a family of linear grammars characterizing $\Psi^{-1}(\mathcal{L})$.*

Our observations are easily generalizable towards regular or linear tuple languages, cf. [25] (which are indeed equivalent to regular or linear simple matrix languages, cf. [9,29,34]) or towards regular-like expressions and their automata, see [7]. We give a combination of Definitions 25 and 26 of Brzozowski as well as of those of Păun, incorporating a suitable "even"-condition, in the following:

Definition 16. *An* even one-sided linear parallel grammar *of* order n *with* direction vector $\delta = \delta_1 \dots \delta_n \in \{L,R\}^n$ *is an* $(n+3)$*-tuple* $G = (V_1, \dots, V_n, \Sigma, M, S)$, *where* $\{S\}, V_1, \dots, V_n, \Sigma$ *are pairwise disjoint alphabets* ($V_N = \bigcup_{i=1}^{n} V_i \cup \{S\}$ *contains the nonterminals and* Σ *the terminals), and* M *is a finite set of matrices of the form*

1. $(S \to A_1 \dots A_n)$, *for* $A_i \in V_i$, $1 \le i \le n$, *or*
2. $(A_1 \to \lambda, \dots, A_{n-1} \to \lambda, A_n \to x_n)$, *for* $A_i \in V_i$, $x_n \in \Sigma^{<n}$, $1 \le i \le n$, *or*
3. $(A_1 \to x_1 B_1 y_1, \dots, A_n \to x_n B_n y_n)$, *for* $A_i, B_i \in V_i$, $x_i = \lambda$ *and* $y_i \in \Sigma$ *iff* $\delta_i = R$, $x_i \in \Sigma$ *and* $y_i = \lambda$ *iff* $\delta_i = L$, *for* $1 \le i \le n$.

Let $V_G = V_N \cup \Sigma$. *For* $x, y \in V_G^*$, *we write* $x \Rightarrow y$ *iff either (i)* $x = S$, $(S \to y) \in M$, *or (ii)* $x = u_1 A_1 v_1 \dots u_n A_n v_n$, $y = u_1 w_1 v_1 \dots u_n w_n v_n$, *and* $(A_1 \to w_1, \dots, A_n \to w_n) \in M$. *As usual, define* $L(G) = \{x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x\}$, *where* $\overset{*}{\Rightarrow}$ *is the reflexive transitive closure of relation* $\Rightarrow$. *Let the corresponding language family be denoted by* δ*-ELPL, where* $\delta \in \{L,R\}^+$.

Remark 17. Obviously, REG = R-ELPL = L-ELPL, and ELL = LR-ELPL = RL-ELPL.[7] Moreover, R^n-ELPL is the class of even equal matrix languages considered by Takada [34], and $(LR)^n$-ELPL is the class of even linear simple matrix languages (of degree n) considered by Fernau [9]. It can be shown that, for every direction vector δ, δ-ELPL $\subseteq (LR)^{2|\delta|}$-ELPL.

It is now a rather easy exercise to extend the notion of uniformly described grammar families for each direction vector within the definition of even one-sided linear parallel grammars. Similarly, Lemma 12 and Theorem 13 transfer to this more general case.

[7] The last equality can be seen by the fact that RL-ELPL grammars can be viewed as an alternative formalization of regularly controlled external contextual grammars, whose relation to linear grammars is exhibited in [28, Section 12.3].

6 Further Inference Methods

6.1 Morphic Generator Grammatical Inference

This methodology has been proposed in [14,16]. Here, the starting point is the well-known fact that every regular language is the image of a 2-testable language under a letter-to-letter morphism [26]. So, the general grammatical inference method as proposed by García *et al.* [16] is the following one: Given an input sample from an unknown (regular) language, choose a mapping g to transform the sample. Then, an inference method for 2-TLSS is applied to the transformed sample in order to obtain a 2-TLSS language. Finally, a morphism h is selected in order to transform the conjectured language to the original regular one. If g and h are well selected, then the target regular language can be learned from the original positive input sample. Unfortunately, there is no way to characterize the mappings g and h in order to perform the learning task. Furthermore, as a consequence of a previous work by Angluin [3], the general strategy to learn regular languages from only positive sample is not possible. Here, we can prove the following generalization to the method by García *et al.*:

Theorem 18. *Let Ψ, Φ be families of permutations. For every $k \geq 1$, every language from $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, REG)$ is the image of a language from $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi,$ 2-$TLSS$) under a letter-to-letter morphism.*

Proof. (Sketch) Consider a language $L \in \mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, \mathrm{REG})$, $L \subseteq \Sigma^*$. This means that $\Psi^{-1}(L) = L(G^{\Sigma}_{0,k}, \Phi(R))$ for a suitable regular language R. By [26], there exists a letter-to-letter morphism $h : X \times \Sigma^{\leq k} \to \Sigma^{\leq k}$ such that $R = h(R')$ for some $R' \in$ 2-TLSS. Since Φ is a permutation, we have $\Phi(R) = \Phi(h(R')) = h(\Phi(R'))$. Every word $(x_1, \alpha_1) \ldots (x_n, \alpha_n) \in X \times \Sigma^{\leq k}$, can be viewed as control word of $G^{X \times \Sigma}_{0,k}$, considering $(X \times \Sigma)^{\leq k}$ as a subset of $X \times \Sigma^{\leq k}$. Taking now the natural projection letter-to-letter morphism $h' : X \times \Sigma \to \Sigma$, we conclude $L = h'(L')$, where $\Psi^{-1}(L') = L(G^{X \times \Sigma}_{0,k}, \Phi(R'))$. □

This suggests the following methodology for identifying languages $L \subseteq \Sigma^*$ from $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, \mathrm{REG})$: [8]

1. Choose a (larger) alphabet Σ' and a letter-to-letter morphism $h : \Sigma' \to \Sigma$.
2. Choose an easily computable function $g : \Sigma^* \to \Sigma'^*$ such that $h(g(w)) = w$ for all $w \in \Sigma^*$.
3. The input sequence $w_1, w_2, \ldots$ is transformed into the sequence $g(w_1), g(w_2), \ldots$ which is given to the identification algorithm for $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi,$ 2-TLSS).
4. The output language sequence $L_1, L_2, \ldots$ hence obtained is interpreted as the language sequence $h(L_1), h(L_2), \ldots$ of languages over Σ.

[8] Again, we assume the transformations induced by Ψ, $\mathcal{G}_{0,k}$ and Φ be be computable in polynomial time.

6.2 Query Learning

In the query learning model introduced by Angluin [5], the inference machine IM plays an active role (in contrast with Gold's model) in the sense that IM interacts with a teacher T. More precisely, at the beginning of this dialogue, IM is just informed about the terminal alphabet Σ of the language L that IM should learn. IM may ask T the following questions:

Membership query Is $w \in L$?
Equivalence query Does the hypothesis grammar G generate L?

Teacher T reacts as follows to the questions:

1. To a membership-query, T answers either "yes" or "no".
2. To an equivalence-query, T answers either "yes" (here, the learning process may stop, since L has performed its task successfully) or "no, I will show you a counterexample w."

Since Angluin showed that all regular languages can be learned in polynomial time by using the learner-teacher dialogue just explained, we can immediately infer:

Theorem 19. *Let $k \geq 1$. If Ψ and Φ are computable in polynomial time, then $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, REG)$ can be learned in the query learning model in polynomial time.* □

Remark 20. When equivalence queries are not reckoned as oracle calls, they should be computable. Since $\Psi(L) = \Psi(L')$ iff $L = L'$, it is not hard to see that equivalence is indeed decidable within the $(\Psi, \mathcal{G}_{0,k}, \Phi)$-setting due to the decidability of the equivalence problem for regular languages.

In the query learning model, the complete power of the formalism is not needed:

Theorem 21. *Let $k \geq 1$. If Ψ and Φ are computable in polynomial time, then $\mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, REG) = \Xi(REG)$ for some polynomial-time computable permutation Ξ.*

Proof. By definition, $L \in \mathcal{L}(\Psi, \mathcal{G}_{0,k}, \Phi, \mathrm{REG})$ if $\Psi^{-1}(L) = L(G^{\Sigma}_{0,k}, \Phi(R))$ for some regular language R. Due to Remark 6, $L(G^{\Sigma}_{0,k}, M) = \tau(M)$ for some rational transduction τ. Moreover, for another permutation Φ', $L(G^{\Sigma}_{0,k}, \Phi(R)) = \Phi'(\tau(R)) = \Phi'(R')$ for some regular language R'. Setting $\Xi = \Psi\Phi'$ yields the theorem. □

This implies that when we are interested in language classes induced by the whole class of regular languages, we can confine ourselves to permutations, which conceptually simplifies all considerations in this case.

7 Conclusions

We presented two quite powerful general and mutually related mechanisms to define further efficiently learnable language classes from already known ones, namely

1. by using families of permutations and
2. by exploiting control language features.

Such "language class generators" can be quite useful since it has turned out that, in various applications, it is necessary to make a choice of the language class to be learned based on experience or additional knowledge inspired by the application, see, e.g., the discussion in [1]. Therefore, it seems to be good if one could enhance the set of possible choices and use, e.g., known structural information like bracket structures.

Furthermore, since polynomial-time computable permutation families are closed under permutationwise composition, it makes sense to consider language families like the class $\Psi^{-1}(\Psi^{-1}(\Psi^{-1}(\mathcal{L})))$ which is also efficiently inferrable from positive data if $\mathcal{L}$ is.

Similarly, any class like $\mathrm{CONTROL}(\mathcal{G}_0, \mathrm{CONTROL}(\mathcal{G}_0, \mathrm{CONTROL}(\mathcal{G}_0, \mathcal{L})))$ is efficiently inferrable from positive data if $\mathcal{L}$ is.

Such hierarchies deserve further studies, as begun in [37,36].

Finally, it would be interesting to develop further general techniques in order to apply known learning (especially, identification) algorithms to other language families. One such technique could consists in a suitable *splitting* of input words, as exhibited in the case of externally contextual languages in [13] and in the case of parallel communicating grammars systems in [12].

References

1. H. Ahonen. *Generating grammars for structured documents using grammatical inference methods*. Phd thesis. Also: Report A-1996-4, Department of Computer Science, University of Helsinki, Finland, 1996.
2. V. Amar and G. Putzolu. On a family of linear grammars. *Information and Control*, 7:283–291, 1964.
3. D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
4. D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982.
5. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
6. J. Berstel. *Transductions and Context-Free Languages*, volume 38 of *LAMM*. Stuttgart: Teubner, 1979.
7. J. A. Brzozowski. Regular-like expressions for some irregular languages. In *IEEE Conf. Record of 9th Ann. Symp. on Switching and Automata Theory SWAT*, pages 278–280, 1968.

8. J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Berlin: Springer, 1989.
9. H. Fernau. Efficient learning of some linear matrix languages. In T. Asano et al., editors, *COCOON'99*, volume 1627 of *LNCS*, pages 221–230, 1999.
10. H. Fernau. Learning of terminal distinguishable languages. Technical Report WSI–99–23, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, 1999. Short version published in the proceedings of AMAI 2000, see `http://rutcor.rutgers.edu/~amai/AcceptedCont.htm`.
11. H. Fernau. k-gram extensions of terminal distinguishable languages. In *Proc. International Conference on Pattern Recognition*. IEEE/IAPR, 2000. To appear.
12. H. Fernau. PC grammar systems with terminal transmission. In *Proc. International Workshop on Grammar Systems*, 2000. To appear.
13. H. Fernau and M. Holzer. External contextual and conditional languages. To appear in a book edited by Gh. Păun, 1999.
14. P. García et al. On the use of the morphic generator grammatical inference (MGGI) methodology in automatic speech recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 4:667–685, 1990.
15. P. García and E. Vidal. Inference of k-testable languages in the strict sense and applications to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:920–925, 1990.
16. P. García, E. Vidal, and F. Casacuberta. Local languages, the successor method, and a step towards a general methodology for the inference of regular grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:841–845, 1987.
17. P. García, E. Vidal, and J. Oncina. Learning locally testable languages in the strict sense. In *First International Workshop on Algorithmic Learning Theory ALT'90*, pages 325–328, 1990.
18. E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
19. J. Gregor. Data-driven inductive inference of finite-state automata. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(1):305–322, 1994.
20. S. A. Greibach. Control sets on context-free grammar forms. *Journal of Computer and System Sciences*, 15:35–98, 1977.
21. T. Head, S. Kobayashi, and T. Yokomori. Locality, reversibility, and beyond: Learning languages from positive data. In *ALT'98*, volume 1537 of *LNCS*, pages 191–204, 1998.
22. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Reading (MA): Addison-Wesley, 1979.
23. E. Mäkinen. The grammatical inference problem for the Szilard languages of linear grammars. *Information Processing Letters*, 36:203–206, 1990.
24. E. Mäkinen. A note on the grammatical inference problem for even linear languages. *Fundamenta Informaticae*, 25:175–181, 1996.
25. H. A. Maurer and W. Kuich. Tuple languages. In W. D. Itzfeld, editor, *Proc. of the ACM International Computing Symposium*, pages 882–891. German Chapter of the ACM, 1970.
26. T. Y. Medvedev. On the class of events representable in a finite automaton (*translated from russian*). In E. F. Moore, editor, *Sequential Machines – Selected papers*, pages 227–315. Addison–Wesley, 1964.
27. M. Nivat. Transductions des langages de Chomsky. *Ann. Inst. Fourier, Grenoble*, 18:339–456, 1968.

28. Gh. Păun. *Marcus contextual grammar*. Studies in Linguistics and Philosophy. Dordrecht: Kluwer Academic Publishers, 1997.
29. Gh. Păun. Linear simple matrix languages. *Elektronische Informationsverarbeitung und Kybernetik (jetzt J. Inf. Process. Cybern. EIK)*, 14:377–384, 1978.
30. V. Radhakrishnan. *Grammatical Inference from Positive Data: An Effective Integrated Approach*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay (India), 1987.
31. V. Radhakrishnan and G. Nagaraja. Inference of regular grammars via skeletons. *IEEE Transactions on Systems, Man and Cybernetics*, 17(6):982–992, 1987.
32. J. Ruiz, S. España, and P. García. Locally threshold testable languages in strict sense: application to the inference problem. In V. Honvar and G. Slutzki, editors, *Grammatical Inference, 4th Intern. Coll. ICGI-98*, volume 1433 of *LNCS*, pages 150–161, 1998.
33. Y. Takada. Grammatical inference of even linear languages based on control sets. *Information Processing Letters*, 28:193–199, 1988.
34. Y. Takada. Learning even equal matrix languages based on control sets. In A. Nakamura et al., editors, *Parallel Image Analysis, ICPIA'92*, volume 652 of *LNCS*, pages 274–289, 1992.
35. Y. Takada. Learning semilinear sets from examples and via queries. *Theoretical Computer Science*, 104:207–233, 1992.
36. Y. Takada. A hierarchy of language families learnable by regular language learning. *Information and Computation*, 123:138–145, 1995.
37. Y. Takada. Learning formal languages based on control sets. In K. P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *LNCS/LNAI*, pages 317–339, 1995.