Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Learning in the limit with lattice-structured hypothesis spaces

Jeffrey Heinz[a], Anna Kasprzik[b], Timo Kötzing[c,*]

[a] *Department of Linguistics and Cognitive Science, University of Delaware, Newark, DE 19716, USA*

[b] *FB IV – Abteilung Informatik, Universität Trier, 54286 Trier, Germany*

[c] *Department 1: Algorithms and Complexity, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany*

A R T I C L E   I N F O

Article history:
Received 1 November 2011
Received in revised form 20 May 2012
Accepted 18 July 2012
Communicated by R. Gavalda

Keywords:
Learning from positive data
Lattice theory

A B S T R A C T

We define a collection of language classes which are TxtEx-learnable (learnable in the limit from positive data). The learners map any data input to an element of a fixed lattice, and keep the least upper bound of all lattice elements thus obtained as the current hypothesis. Each element of the lattice is a grammar for a language, and the learner climbs the lattice searching for the right element. We call these classes in our collection *lattice classes*.

We provide several characterizations of lattice classes and their learners, which suggests they are very natural. In particular, we show that any class of languages is a lattice class iff it is TxtEx-learnable consistently, conservatively, set-drivenly, and strongly monotonically.

We show several language classes previously discussed in the literature to be lattice classes, including the locally k-testable classes, the piecewise k-testable classes, the k-reversible languages and the pattern languages. We also show that lattice classes contain three previously known collections of language classes: string extension language classes, function-distinguishable language classes, and closed-set systems.

Finally, the lattice perspective helps analyze the learning of these classes. Illustrations include query-learning results in dependence on the lattice structure, characterizations of closure properties and the VC-dimension of lattice classes in terms of lattice properties.

1. Introduction

The problem of generalizing from data to patterns is an important one in computer science, artificial intelligence, linguistics, robotics, and many other fields. In particular, the problem is central to subfields of computer science such as machine learning [35] and grammatical inference [13].

An insightful learning paradigm is *Gold-style learning* [20]. In Gold-style learning, a learner is given increasingly many *positive examples* taken from a fixed, computably enumerable *target language* $L \subseteq \Sigma^*$ where Σ^* denotes the set of all finite-length words over a finite alphabet Σ. After each example, the learner *conjectures* a hypothesis, or description of L.[1] If, on all enumerations of all and only the elements of L, the learner outputs in the limit a conjecture which correctly describes L, then we say the learner *TxtEx-learns* L. If a learner can TxtEx-learn every language in a class of languages $\mathcal{L}$, we say the learner TxtEx-learns $\mathcal{L}$. TxtEx-learning is also known as identification in the limit from positive data.

Gold [20] showed that no superfinite class of languages—i.e. any class with all finite languages and at least one infinite language—is TxtEx-learnable. A significant consequence of this result was that no major class of formal languages, such as

* Corresponding author.
E-mail addresses: heinz@udel.edu (J. Heinz), kasprzik@informatik.uni-trier.de (A. Kasprzik), timo.koetzing@mpi-inf.mpg.de, koetzing@mpi-inf.mpg.de (T. Kötzing).

[1] For example, context-free grammars could be used as descriptions for context-free languages.

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.07.017

the regular languages, is TxtEx-learnable. However, Angluin [3] provided a characterization of TxtEx-learnable classes and many interesting classes of languages are continually being discovered [2,4,22,38,54,9]. Lang et al. [34] provide a recent overview of learning from positive data (readers can consult [28,13] for treatments of this and other learning paradigms). Additionally, much research has also been devoted to studying general properties of large, natural groups of TxtEx-learnable classes [53,33,46,16,27]. This article belongs to this last-mentioned research effort.

We present a collection of language classes, called *Lattice Classes (LCs)*. The main contribution of this paper is to show the following about LCs.

(i) LCs are TxtEx-learnable by learners with many desirable properties: the learners are incremental, consistent, conservative, set-driven, and strongly monotonic (Theorem 2.9).
(ii) LCs have multiple characterizations (Theorems 4.6 and 4.7).
(iii) LCs generalize three other TxtEx-learnable collections of language classes (function-distinguishable classes [16], string extension classes [23,25], and closed-set systems [12]).
(iv) LCs provide a unified, insightful learning-theoretic framework for the analysis of several important classes of languages, including the locally k-testable classes, the piecewise k-testable classes, the k-reversible languages and the pattern languages, and many others.

Note that the **TxtEx**-learnability of lattice classes follows straightforwardly from the lattice structure, as do many of the desirable properties named in item (i) in the list above. We consider the simplicity and consequent generality of this approach a strength of our analysis.

A related approach which uses lattices for learning from positive data can be found in [31,30] and related papers, where the lattice structure of the hypothesis space is analyzed in the setting of *approximate identification.*

We define any LC in terms of a lattice V and a function f mapping strings to elements of V. We use lattices as defined by Birkhoff, [8]; essentially, a lattice is a partially ordered set with additional properties (see Definition 2.1). Each element $v \in V$ corresponds to the set of all strings mapped to or below v by f. As we assume f and the lattice order of V to be computable, any LC is uniformly decidable (see Theorem 4.5(i)).

We give learners for these classes which proceed by mapping strings to nodes and then climbing the lattice by only keeping track of the least upper bound as the current hypothesis. We call any learners of this kind *Lattice Learners (LLs)*.

Section 2 makes Lattice Learning precise. We further analyze Lattice Learners and Lattice Classes in Section 4. In particular, we give *two* insightful characterizations of LLs in Theorem 4.6, and *three* characterizations of LCs in Theorem 4.7. This establishes the LCs as very naturally arising learnable language classes.

Sections 3 and 5 provide simple and more complex examples of lattice classes, respectively.

We address *query learning* [5] of Lattice Classes and special cases thereof in Section 6. We discuss complexity issues and show that Lattice Classes can be learned in a particularly straightforward way from *equivalence queries*.

Finally, Section 7 considers the VC Dimension of a lattice classes; we give a characterization and a (weak) upper bound on the VC Dimension of lattices with finite width. The VC-dimension plays an important role in PAC-learning.

Familiarity with lattice theory is useful to understand this paper, but not completely necessary. For introductions to lattice theory, the reader is referred to the textbooks [8] (a classic) and [37] (available online).

2. Definitions and basic properties

Any unexplained complexity-theoretic notions are from [42]. All unexplained general computability-theoretic notions are from [44].

The symbol $\mathbb{N}$ denotes the set of natural numbers, $\{0, 1, 2, \ldots\}$. We let Σ be a countable alphabet (a non-empty countable set; we allow for countably infinite alphabets), and Σ^* denotes the set of all finite words over Σ. A *language* is any set $L \subseteq \Sigma^*$. For each k, Σ^k denotes the set of all words of length exactly k. We denote the empty word by ε, the length of word x by $|x|$, and the reversal of a word x by *reverse*(x).

For any set A, $|A|$ denotes the cardinality of A. For sets A, B, we let $A \setminus B = \{a \in A \mid a \notin B\}$, and $\overline{A}$ be the complement of A; with $\mathrm{Pow}(A)$ ($\mathrm{Pow}_{\mathrm{fin}}(A)$) we denote the set of all (finite) subsets of A.

The quantifier $\forall^\infty x$ means "for all but finitely many x", the quantifier $\exists^\infty x$ means "for infinitely many x".

A partition π of a set A is a collection of pairwise disjoint non-empty subsets of A whose union equals A. These subsets are called *blocks*, and the block of partition π containing $a \in A$ is denoted $[a]_\pi$. A binary relation R over a set A is said to be an equivalence relation iff it is reflexive, transitive, and symmetric. Now R naturally induces a partition over A: for all $a \in A$, the set of all b such that aRb is called the *equivalence class* of a. Furthermore, the set of all these equivalence classes are the blocks of a partition. For a function f, we let $\mathrm{dom}(f)$ and $\mathrm{range}(f)$ denote, respectively, the domain and range of f. We sometimes denote a function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n \centerdot f(x_1, \ldots, x_n)$. For example, with $c \in \mathbb{N}$, $\lambda x \centerdot c$ is the constant c function of one argument.

A function ψ is *partial computable* iff there is a deterministic Turing machine computing ψ. $\mathcal{P}$ and $\mathcal{R}$ denote, respectively, the set of all partial computable and the set of all total computable functions $\mathbb{N} \to \mathbb{N}$. We say that ψ is *polytime* iff ψ is computable on some Turing machine in a number of steps polynomial in the length of the input. Whenever we consider (partial) computable functions on objects like finite sequences, finite words, or finite sets, we assume those objects to be efficiently coded as natural numbers (so we can speak of the functions belonging to $\mathcal{P}$ and $\mathcal{R}$). The size of any such finite object is the size of its code number. If a function f is defined for $x \in \mathbb{N}$ we write $f(x)\downarrow$, and we say that f *converges on* x.

We fix a computable function φ such that, for all $p, x \in \mathbb{N}$, $\varphi(p, x) = \varphi_p(x)$ is the output of the Turing-machine coded by p when given x as input (and undefined, if the Turing-machine does not terminate or if p does not correspond to a Turing-machine).

For all p, W_p denotes the computably enumerable (ce) set $\mathrm{dom}(\varphi_p)$.

Note that, for infinite alphabets, the size of words of length 1 is unbounded.

After these general definitions, we will now turn to definitions more specific to this paper. First we introduce lattices and *Lattice Spaces* and then show how we use them for learning.

2.1. Lattice spaces

Definition 2.1. Let V be a non-empty set and $\sqsubseteq$ a binary relation over V. The pair $(V, \sqsubseteq)$ is a *partially ordered set* iff $\sqsubseteq$ is anti-symmetric, reflexive and transitive. We use the symbols $\sqsubset$, $\sqsupseteq$ and $\sqsupset$ as usual (to denote “properly below,” “above” and “properly above”).

Let $(V, \sqsubseteq)$ be a partially ordered set. For any set $S \subseteq V$, $v \in V$ is called

- an *upper bound of S* iff $\forall a \in S : a \sqsubseteq v$;
- a *lower bound of S* iff $\forall a \in S : v \sqsubseteq a$;
- a *maximum of S* iff v is the upper bound of S and $v \in S$;
- a *minimum of S* iff v is the lower bound of S and $v \in S$;
- a *least upper bound* or *supremum of S* iff v is the minimum of the set of upper bounds of S;
- a *greatest lower bound* or *infimum of S* iff v is the maximum of the set of lower bounds of S;

Note that, for a given set, there is at most one supremum and at most one infimum. If V has a minimum element, we denote it by $\bot_V$, a maximum element by $\top_V$. $(V, \sqsubseteq)$ is called

- an *upper semi-lattice* iff each two elements of V have a supremum;
- a *lower semi-lattice* iff each two elements of V have an infimum;
- a *lattice* iff each two elements of V have both a supremum and an infimum.

The supremum of two elements $a, b \in V$ is denoted by $a \sqcup b$ if it exists, and the infimum of two elements $a, b \in V$ is denoted by $a \sqcap b$, if it exists. For all sets D, we use $\bigsqcup D$ to denote the supremum of D and $\bigsqcap D$ to denote the infimum of D, if they exist. Note that, in an upper (lower) semi-lattice, each non-empty finite set has a supremum (infimum), which equals the iterated supremum (infimum) of its elements, as the binary supremum (infimum) is an associative and commutative operation. If V has a minimum element, then, by convention, $\bigsqcup \emptyset = \bot_V$. Likewise, if V has a maximum element, we let $\bigsqcap \emptyset = \top_V$.

For two partially ordered sets U, V, a function $h : U \to V$ is called an *order embedding* iff, for all $a, b \in U$, $a \sqsubseteq_U b \Leftrightarrow h(a) \sqsubseteq_V h(b)$. An *order isomorphism* is a bijective order embedding.

Let V be a partially ordered set with a minimum element. An element $a \in V$ is called an *atom* iff $a \neq \bot_V$ and the set $\{b \mid \bot_V \sqsubseteq b \sqsubseteq a\}$ equals the set $\{\bot_V, a\}$. A lattice is called *complete* iff all sets of lattice elements (including infinite sets) have a supremum and an infimum. A lattice is called *boolean* iff V has both a minimal element $\bot$ and a maximal element $\top$ and for all $a \in V$, there exists $\overline{a} \in V$ such that $a \sqcap \overline{a} = \bot$ and $a \sqcup \overline{a} = \top$.

A sequence $(a_n)_{n\in\mathbb{N}}$ is called an *infinitely ascending chain* iff, for all i, $a_i \sqsubset a_{i+1}$. A subset $U \subseteq V$ is called an *anti-chain* iff no two elements of U are comparable.

Example 2.2. For each $k \in \mathbb{N}$ for example, $(\mathrm{Pow}(\Sigma^k), \subseteq)$ is a lattice. This lattice is the set of all finite sets, including the empty set, that contain only words of length k, with inclusion as the order. We call this lattice $V_{\mathrm{Pow}(\Sigma^k)}$. Fig. 1 shows $V_{\mathrm{Pow}(\Sigma^2)}$ with $\Sigma = \{a, b\}$.

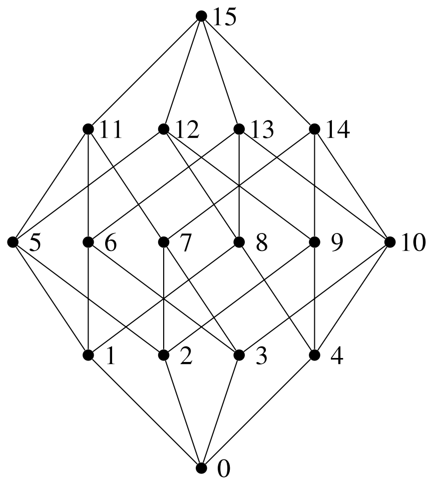
Now we are ready to introduce the central concepts of *Lattice Spaces* and *Lattice Classes*.

Definition 2.3. For an upper semi-lattice V and a function $f : \Sigma^* \to V$ such that f and $\sqcup$ are (total) computable, (V, f) is called a *Lattice Space (LS)*, iff, for each $v \in V$, there exists a finite $D \subseteq \mathrm{range}(f)$ with $\bigsqcup D = v$.[2]

We call two LS (V, f) and (U, g) *isomorphic* iff there is an upper semi-lattice isomorphism α from V to U such that, for all $x \in \Sigma^*$, $\alpha(f(x)) = g(x)$.

(V, f) is called *polytime* iff f and suprema in V are polytime.

[2] This definition might seem a little strange at first, and in general one could define lattice spaces without those restrictions. However, elements that are not the finite suprema of elements from range(f) are not directly useful for our purposes, and many of our theorems would have to be stated in terms of the “stripped” sub semi-lattice one gets from restricting to all elements which are finite union of elements from range(f). Thus, for notational purposes, we only allow for “stripped” lattice spaces in the first place.



node	set	node	set
15	$\{aa, ab, ba, bb\}$	7	$\{ab, ba\}$
14	$\{ab, ba, bb\}$	6	$\{aa, ba\}$
13	$\{aa, ba, bb\}$	5	$\{aa, ab\}$
12	$\{aa, ab, bb\}$	4	$\{bb\}$
11	$\{aa, ab, ba\}$	3	$\{ba\}$
10	$\{ba, bb\}$	2	$\{ab\}$
9	$\{ab, bb\}$	1	$\{aa\}$
8	$\{aa, bb\}$	0	$\emptyset$

Fig. 1. The lattice $V_{\text{Pow}(\Sigma^2)}$ with $\Sigma = \{a, b\}$.

Example 2.4. Continuing Example 2.2, for fixed k, consider the function that maps words to the set of their *k-factors* — any substring of length k of word w is a *k-factor* of w. In other words, for each k, let $fac_k : \Sigma^* \rightarrow V_{\text{Pow}(\Sigma^k)}$ be such that $x \mapsto \{v \in \Sigma^k \mid \exists u, w \in \Sigma^* : x = uvw\}$. Then $(V_{\text{Pow}(\Sigma^k)}, fac_k)$ is an LS. To illustrate, fac_2 maps word *abb* to node 9 in Fig. 1, maps *abbaa* to node 15, and maps all words of length less than 2 to node 0.

Clearly, fac_2 naturally defines an equivalence relation such that words are equivalent iff they have the same sets of 2-factors. More generally, for any LS (V, f), the function f defines an equivalence relation: $\forall w, u \in \Sigma^* : w \sim u$ iff $f(w) = f(u)$. This equivalence relation naturally partitions all logically possible words into blocks, which are in one-to-one correspondence with the nodes in V.

Definition 2.5. Let (V, f) be an LS.

- A *grammar* is any $v \in V$.[3]
- The *language of grammar* v is $L_f(v) = \{w \in \Sigma^* \mid f(w) \sqsubseteq v\}$.
- The *class of languages* obtained by all possible grammars is

$$\mathcal{L}_{(V,f)} = \{L_f(v) \mid v \in V\}.$$

Note that we will always consider a fixed lattice associated with any given function f, so we can write $\mathcal{L}_f$ instead of $\mathcal{L}_{(V,f)}$ without any ambiguity.

Any class of languages $\mathcal{L}$ such that there is an LS (V, f) with $\mathcal{L} = \mathcal{L}_f$ is called a *Lattice-structured Class (LC)*.[4] We use LC to denote the set of all Lattice-structured Classes. Further, we omit the subscript f only if it is clear from context.

Example 2.6. Continuing the example begun in Fig. 1, what are the grammars and languages of $(V_{\text{Pow}(\Sigma^2)}, fac_2)$? The grammars are the nodes; i.e. the finite subsets, which represent the *admissible* 2-factors. For example, consider $G = \{aa, ab\}$ (node 5). The language of this grammar is all words which fac_2 maps to a node less than or equal to the grammar in the lattice structure. In other words, a word w belongs to $L(G)$ iff fac_2 maps w to node 5, 2, 1 or 0; i.e. iff the 2-factors of w are a subset of G. Consequently, $L(G)$ contains all and only those words which do not contain any *forbidden* factors (those 2-factors not in G, here *ba* and *bb*). We call the languages in this LS 2-*factor languages*.

More generally, the k-factor languages can also be described in terms of the partition fac_k induces over Σ^*. Although each node v corresponds to one such block, the language of v includes exactly the set of all the words in this block and the words in all blocks corresponding to nodes ordered below v. For example, while the block of the partition corresponding to node 15 only includes words with all four 2-factors, the language of node 15 is the union of all the blocks; i.e. Σ^*.

In [25], *String Extension Classes* are defined, which are a special case of Lattice Classes: String Extension Classes are exactly those Lattice Classes based on the lattice of all finite subsets of a finite set A with inclusion. Further, we get the special case of closed-set systems [12] if we take all *complete* lattices. We give a connection of Lattice Classes to Function Distinguishable Classes in Theorem 5.6.

[3] Note that we assume our grammars to be finite with respect to a relevant measure, i.e., containing for example a finite number of admissible substrings, or other rules.

[4] In formal language theory, several descriptions may define the same language. Observe that for the language classes defined here this is not the case – we have $L_f(u) \neq L_f(v)$ for any two elements $u, v \in V$ with $u \neq v$. See Theorem 4.1.

2.2. Learning in the limit

In this section we define the learning criterion *learning in the limit from positive data* following [20], which forms the basis of our first analysis of our lattice learners.

By $\mathbb{S}\mathrm{eq}$ we denote the set of finite sequences over $\Sigma^* \cup \{\#\}$, where # is a special symbol called "pause". We denote the empty sequence by $\emptyset$. For a sequence $\sigma \in \mathbb{S}\mathrm{eq}$, we let $\mathrm{len}(\sigma)$ denote the length σ, and, for all $i < \mathrm{len}(\sigma)$ we let $\sigma(i)$ be the $i+1$-th element of σ. Concatenation on sequences is denoted by $\diamond$. For all finite or infinite sequences σ over $\Sigma^* \cup \{\#\}$ we let $\mathrm{content}(\sigma) = \{x \in \Sigma^* \mid \exists i < \mathrm{len}(\sigma) : \sigma(i) = x\}$.

Let $L \subseteq \Sigma^*$ and $T : \mathbb{N} \to \Sigma^* \cup \{\#\}$. T is called a *text for L* iff $\mathrm{content}(T) = L$. For any text T and any k, let $T[k]$ denote the sequence of the first k elements of T (in particular, $T[0]$ is the empty sequence). With $\mathbf{Txt}(L)$ we denote the set of all texts for a language L.

For this paper, a *learner* is any total computable function $h : \mathbb{S}\mathrm{eq} \to \mathbb{N}$. We assume the outputs of h to be mapped by a function $\lambda x \centerdot L(x)$ to a language. Whenever no concrete function $\lambda x \centerdot L(x)$ is given, we assume the general-purpose mapping $\lambda x \centerdot W_x$.

A learner h is said to **TxtEx**-*learn* a language L with respect to $\lambda x \centerdot L(x)$ iff, for each text T for L, there is $k \in \mathbb{N}$ such that

(i) $L(h(T[k])) = L$; and
(ii) for all $k' \geq k$, $h(T[k']) = h(T[k])$.

For the minimum such k, we then say that h on T has *converged* after k steps, and denote this by $\mathbf{Conv}(h, T) = k$.

We denote the set of all languages **TxtEx**-*learned* by a learner h with $\mathbf{TxtEx}(h)$. We say that a class of languages $\mathcal{L}$ is **TxtEx**-*learned* (possibly with certain properties) iff there is a learner h (obeying those properties) which **TxtEx**-learns every language in $\mathcal{L}$. Furthermore, we say that h learns a class of languages using a uniformly decidable hypothesis space iff there is a total function returning 1 on all and only the x such that $x \in L(p)$.

The following learner properties have been studied in the literature.

Definition 2.7. Let a learner $h : \mathbb{S}\mathrm{eq} \to \mathbb{N}$ be given. We call h

- iterative [17,52], iff there is a function $h^{it} : \mathbb{N} \times \Sigma^* \to \mathbb{N}$ such that $\forall \sigma \in \mathbb{S}\mathrm{eq}, w \in \Sigma^* : h^{it}(h(\sigma), w) = h(\sigma \diamond w)$;
- polytime iterative, iff there is such a polytime function h^{it};
- set-driven [51,28], iff there is a function $h^{set} : \mathrm{Pow}_{\mathrm{fin}}(\Sigma^*) \to \mathbb{N}$ such that $\forall \sigma \in \mathbb{S}\mathrm{eq} : h^{set}(\mathrm{content}(\sigma)) = h(\sigma)$;
- globally consistent [6], iff $\forall \sigma \in \mathbb{S}\mathrm{eq} : \mathrm{content}(\sigma) \subseteq L(h(\sigma))$;
- locally conservative [3], iff $\forall \sigma \in \mathbb{S}\mathrm{eq}, x \in \Sigma^* : h(\sigma) \neq h(\sigma \diamond x) \Rightarrow x \notin L(h(\sigma))$;
- strongly monotone [26], iff $\forall \sigma \in \mathbb{S}\mathrm{eq}, x \in \Sigma^* : L(h(\sigma)) \subseteq L(h(\sigma \diamond x))$;
- prudent [39], iff $\forall \sigma \in \mathbb{S}\mathrm{eq} : L(h(\sigma)) \in \mathbf{TxtEx}(h)$;
- optimal [20], iff, for all learners h' with $\mathbf{TxtEx}(h) \subseteq \mathbf{TxtEx}(h')$,

$$\begin{gathered}\exists L \in \mathbf{TxtEx}(h), T \in \mathbf{Txt}(L) : \mathbf{Conv}(h', T) < \mathbf{Conv}(h, T)\\ \Rightarrow\\ \exists L \in \mathbf{TxtEx}(h), T \in \mathbf{Txt}(L) : \mathbf{Conv}(h, T) < \mathbf{Conv}(h', T).\end{gathered}$$

Informally, *iterative* learners are those whose next hypothesis only depends on the current data point and the previous hypothesis, and *polytime iterative* learners are iterative learners who can compute the next hypothesis from the previous one and current data point in polynomial time. *Set-driven* learners are ones which are insensitive to the order in which the data points are presented. *Globally-consistent* learners are ones whose current hypothesis always includes all data points observed so far, whereas *locally-conservative* learners are ones which only change a hypothesis h if the current data point is not consistent with h. Learners are *strongly-monotone* provided languages of subsequent hypotheses are always supersets of prior ones. A *Prudent* learner ϕ only ever considers hypotheses which correspond to languages belonging to the class learnable by ϕ. A learner ϕ is *optimal* if there is no other learner for the class which converges more quickly on every language in the class. Note that this notion of optimality is a notion of *pareto optimality*, i.e., improvement in one place would necessarily imply a worsening elsewhere.

2.3. Learning with lattices

For any LS (V, f), we define a learner ϕ_f such that

$$\forall \sigma : \phi_f(\sigma) = \bigsqcup_{x \in \mathrm{content}(\sigma)} f(x).$$

Essentially, this learner maps each data point to the lattice with f and calculates the greatest lower bound. We call ϕ_f a *Lattice Learner (LL)*.

Table 1
Illustrating a Lattice Learner. Fields labeled "Node" refer to Fig. 1.

i	$t(i)$	$t[i]$	$fac_2(t(i))$	Node	$\phi_{fac_2}(t[i])$	Node
0	ab	$\langle ab\rangle$	$\{ab\}$	2	$\{ab\}$	2
1	abb	$\langle ab, abb\rangle$	$\{ab, bb\}$	9	$\{ab, bb\}$	9
2	aba	$\langle ab, abb, aba\rangle$	$\{ab, ba\}$	7	$\{ab, bb, ba\}$	14
3	bba	$\langle ab, abb, aba, bba\rangle$	$\{ba, bb\}$	10	$\{ab, bb, ba\}$	14
…						

Example 2.8. Table 1 illustrates lattice learning of the language L which forbids the 2-factor aa with ϕ_{fac_2}. In other words $L = \{w \in \Sigma^* \mid aa$ is not a 2-factor of $w\}$. This language belongs to the lattice-structured class $\mathcal{L}_{fac_2}$. The learner first observes ab, which fac_2 maps to $\{ab\}$, which corresponds to node 2 in Fig. 1. Upon observing the word abb, ϕ_{fac_2} hypothesizes grammar $G = \{ab, bb\}$, node 9 in Fig. 1. The learner automatically infers that other words belong to the target language. For example, words with the same set of 2-factors belong to the target language, such as $abbb$. Furthermore, words for which the set of 2-factors is a subset of G also belong to the target language, such as bb (since node 4 is ordered below node 9). The learner next observes aba, which fac_2 maps to grammar $\{ab, ba\}$, node 7. By definition, the learner's hypothesis becomes the least upper bound of nodes 7 and 9, which is $\{ab, ba, bb\}$, node 14, and the learner can infer that all words in all blocks associated with all nodes less than or equal to this one belong to the language. Consequently it infers that words like bba (node 10) are in the language as well. Therefore, when $i = 3$ and the learner observes bba, the learner's hypothesis does not change. Assuming all the words are drawn from L, it follows that the learner has converged to the target language on the text given in this example at $i = 2$.

Notice the learner ϕ_f makes inferences in three ways. First, for each word w observed, the learner can generalize automatically to all words in the block associated with the grammar $f(w)$. Second, for each word w observed, the learner can generalize to all words in all blocks corresponding to grammars ordered below $f(w)$. Third, for two words w and v, learners can generalize to the block of words corresponding to $f(w) \sqcup f(v)$ (and by the second method, all blocks corresponding to nodes below $f(w) \sqcup f(v)$). It is the third method by which the learner climbs the lattice structure.

Lattice Learners have all of the desirable properties presented in Definition 2.7.

Theorem 2.9. *Let (V, f) be an LS. Then ϕ_f* **TxtEx***-learns $\mathcal{L}_f$*

(i) *iteratively;*
(ii) *if (V, f) is a polytime LS, polytime iteratively;*
(iii) *set-drivenly;*
(iv) *globally consistently;*
(v) *locally conservatively;*
(vi) *strongly monotonically;*
(vii) *prudently; and*
(viii) *optimally.*

Proof. Regarding **TxtEx**-learnability: Let $L \in \mathcal{L}_f$ and let $v \in V$ be such that $L(v) = L$. Let T be a text for L. As (V, f) is an LS, let $D \subseteq \Sigma^*$ such that $v = \bigsqcup_{x\in D} f(x)$. Thus, $D \subseteq L = \text{content}(T)$. Let k be such that $D \subseteq \text{content}(T[k])$. Then, obviously, $\forall k' \geq k : \phi_f(T[k']) = v$. Regarding the different items of the list, we have:

(i) We let $\phi_f^{it} \in \mathcal{P}$ be such that

$$\forall v, x : \phi_f^{it}(v, x) = \begin{cases} v, & \text{if } x = \#; \\ v \sqcup f(x), & \text{otherwise.} \end{cases} \tag{1}$$

(ii) Clearly, ϕ_f^{it} from (i) is polytime, if (V, f) is a polytime LS.
(iii) Let $\phi^{set} \in \mathcal{P}$ be such that $\forall D : \phi^{set}(D) = \bigsqcup_{x\in D} f(x)$.
(iv) Let σ be a sequence in Σ^*, let $v = \phi_f(\sigma)$ and $x \in \text{content}(\sigma)$. Then $f(x) \sqsubseteq \bigsqcup_{y\in\text{content}(\sigma)} f(y) = \phi_f(\sigma) = v$. Thus, $x \in L(v)$.
(v) Let $\sigma \in \mathbb{S}\text{eq}$ and $x \in \Sigma^*$ with $\phi_f(\sigma) \neq \phi_f(\sigma \diamond x)$. Thus, $\phi_f(\sigma) \neq \phi_f(\sigma) \sqcup f(x)$, in particular, $f(x) \not\sqsubseteq \phi_f(\sigma)$. Therefore, $x \notin L(\phi_f(\sigma))$.
(vi) Let $\sigma \in \mathbb{S}\text{eq}$ and $x \in \Sigma^*$. Clearly, $\phi_f(\sigma) \sqsubseteq \phi_f(\sigma \diamond x)$. Thus,

$$\begin{aligned} L_f(\phi_f(\sigma)) &= \{w \in \Sigma^* \mid f(w) \sqsubseteq \phi_f(\sigma)\} \\ &\subseteq \{w \in \Sigma^* \mid f(w) \sqsubseteq \phi_f(\sigma \diamond x)\} \\ &= L_f(\phi_f(\sigma \diamond x)). \end{aligned}$$

(vii) Prudence is clear, as, for all $\sigma \in \mathbb{S}\text{eq}$ and $x \in L_f(\phi_f(\sigma))$, we have $f(x) \sqsubseteq \phi_f(\sigma)$. Hence, for all texts T for $L_f(\phi_f(\sigma))$, ϕ_f on T will converge to $\phi_f(\sigma)$.

(viii) Optimality follows from consistency, conservativeness and prudence, as stated in [39, Proposition 8.2.2A]. □

3. Simple examples of lattice classes

In this section, we show that four major classes in the subregular hierarchies—the Locally k-Testable, Piecewise k-Testable, and their Strict counterparts [36,47,43]—are sets of lattice classes. These examples are chosen for their cognitive and linguistic interest [45,24].

We already came across the example of k-factor languages (Example 2.6 and its LS $(V_{\text{Pow}(\Sigma^k)}, fac_k)$. A class of languages that makes use of the same lattice $V_{\text{Pow}(\Sigma^k)}$ with a function other than fac_k is the class of *k-subsequence languages*. A string $v = a_1 \dots a_k$ is a *subsequence* of string w ($v \sqsubseteq_{ssq} w$) iff there exist $u_0 \dots u_k \in \Sigma^*$ such that $w = u_0 a_1 u_1 \dots a_k u_k$. Then, for each k, let $ssq_k : \Sigma^* \to V_{\text{Pow}(\Sigma^k)}$ such that $x \mapsto \{v \in \Sigma^k \mid v \sqsubseteq_{ssq} x\}$. Then $(V_{\text{Pow}(\Sigma^k)}, ssq_k)$ is an LS. In the same way that k-factor languages forbid certain factors, k-subsequence languages forbid certain subsequences.

Example 3.1. The language corresponding to $\{aa, ab, ba\}$ (node 11 in Fig. 1) is $\Sigma^*/\Sigma^*b\Sigma^*b\Sigma^*$; i.e. all words which do not contain the subsequence bb. It follows that the lattice learner for this class converges to this pattern after observing words aa, ab, ba (since it has seen all the allowable 2-long subsequences).

The $(V_{\text{Pow}(\Sigma^k)}, ssq_k)$ and $(V_{\text{Pow}(\Sigma^k)}, fac_k)$ Lattice Spaces show that an LS (V, f) separates the structure of the class V from its semantics (which is given by f) since the k-factor languages and the k-subsequence languages have the same lattice structure but differ only with respect to the function f.

The k-subsequence language classes are very closely related to the Locally k-Piecewise Languages in the Strict Sense (Strictly k-Piecewise, k-SP) [43]. Rogers et al. show that one characterization of the class of all languages that are k-SP (for some k) is that they are precisely the class of all languages closed under a subsequence. Although very similar, the k-subsequence languages are not exactly the k-SP class. To illustrate, let $\Sigma = \{a, b, c\}$ and consider the k-subsequence language L defined by the permissible subsequences $\{aa, ab, ba\}$ (so all other 2-subsequences are forbidden). Since $ssq_k(c) = \emptyset$ it must belong to L. Both L and $L/\{c\}$ are closed under a subsequence but only L is a k-subsequence language.

Nonetheless, the k-SP languages are a lattice class. Let $ssq_{\leq k}$ map words to all their subsequences of length less than or equal to k. Let $V_{k\text{SP}} = \{ssq_{\leq k}(w) \mid w \in \Sigma^*\}$. $V_{k\text{SP}}$ under inclusion is a lattice and $(V_{k\text{SP}}, ssq_{\leq k})$ is a lattice space. Furthermore, the class of languages given by this lattice space is exactly k-SP.

The Piecewise k-Testable (k-PT) languages [47] are the minimal collection of languages which includes the languages obtained by closing the k-SP languages under boolean operations [43]. Equivalently, this is the class of languages where two words with the same set of k-subsequences either both belong to the language or both do not [21]. Let $V_{k\text{PT}}$ be the lattice whose nodes are the powerset of the set of nonempty nodes in $V_{ssq_{\leq k}}$ and let inclusion be the partial order (so the atoms of the lattice are the nonempty nodes themselves). Let $f_{k\text{PT}}(w)$ return a singleton set whose sole element is $ssq_{\leq k}(w)$. Then $(V_{k\text{PT}}, f_{k\text{PT}})$ is the lattice space which accepts the k-PT languages.

Example 3.2. Consider the 2-PT languages and observe that $f_{kPT}(ab) = \{\{\epsilon, a, b, ab\}\}$ and $f_{kPT}(ba) = \{\{\epsilon, a, b, ba\}\}$. So if the lattice learner observes ab and then ba, the current grammar would be node $\{\{\epsilon, a, b, ab\}, \{\epsilon, a, b, ba\}\}$ in $V_{2\text{PT}}$.

Similarly, the k-factor languages are closely related to the Locally k-Testable in the Strict Sense (Strictly k-Local, k-SL) languages [36]. k-SL classes make distinctions not just on the basis of substrings but instead on the basis of interior k-factors, and the length $k-1$ prefix and suffix. In order to establish that k-SL is a lattice space we need to establish that any two Lattice Spaces can be 'multiplied' together to create a new lattice space.

Consider two lattice spaces, (V_1, f_1) and (V_2, f_2), which define classes $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. These determine a new lattice space (V, f) where $V = V_1 \times V_2$ (and $(v_1, v_2) \sqsubseteq (v_1', v_2')$ iff $v_1 \sqsubseteq_1 v_1'$ and $v_2 \sqsubseteq_2 v_2'$), and $f = (f_1, f_2)$. It follows straightforwardly that the language class $\mathcal{L}$ determined by (V, f) is

$$\mathcal{L} = \{L_1 \cap L_2 \mid L_1 \in \mathcal{L}_1 \text{ and } L_2 \in \mathcal{L}_2\}.$$

We write $(V_1, f_1) \times (V_2, f_2) = (V, f)$. Note that $\times$ is associative (up to isomorphism).

This result is an interesting complement to [53], which shows that the finite *unions* of languages drawn from **TxtEx**-learnable classes are also **TxtEx**-learnable. In contrast, the above establishes that the *intersections* of languages drawn from lattice classes are also **TxtEx**-learnable. (Wright's results are extended in [46,30].)

Returning to the k-SL languages, let the (left-edge) prefix of length k, the (right-edge) suffix of length k, and the interior k-factors of a word w be

$$L_k(w) = \{u \in \Sigma^k \mid \exists v \in \Sigma^* \text{ such that } w = uv\};$$
$$R_k(w) = \{u \in \Sigma^k \mid \exists v \in \Sigma^* \text{ such that } w = vu\};$$
$$I_k(w) = fac_k(w).$$

Then $(V_{\text{Pow}_{\text{fin}}(\Sigma^{k-1})}, L_{k-1})$, $(V_{\text{Pow}_{\text{fin}}(\Sigma^{k-1})}, R_{k-1})$, and $(V_{\text{Pow}_{\text{fin}}(\Sigma^k)}, I_k)$ are all lattice spaces. The product of these lattice spaces yields a lattice space which describes the k-SL class. We refer to this lattice space as $(V_{k\text{LRI}}, LRI_k)$.

Similar to the k-PT languages, the class of Locally k-Testable (k-LT) languages is the minimal collection of languages which includes the languages obtained by closing the k-SL languages under boolean operations [36]. Similar to the Piecewise k-Testable Languages, the lattice space for the Locally k-Testable Languages is $(V, f) = (\mathrm{Pow}(V_{LRI_k}/\emptyset), w \mapsto \{LRI_k(w)\}$, with the lattice itself being ordered, as before, by inclusion.

In this section we established that four important subregular classes are all lattice classes (and thus learnable by lattice learners). Along the way, we showed how lattice classes can be combined to yield new ones. This provides a certain flexibility to learn multiple patterns of different types that may be present in data. For example, if both local dependencies and certain kinds of long-distance dependencies need to be learned in some domain, this can be accomplished with a Lattice Learner invoking the function $(fac_k, ssq_{k'})$, for example. Learners of these types resemble what cognitive scientists call *modular learning* [19].

Many more examples like the ones mentioned here can be found in [25]. However, the lattice formulation developed here is stronger than the results there. We turn to more complex examples in Section 5 which demonstrate the full power of the present proposal. But first, we establish important properties of lattice learners.

4. Properties of lattice learning

In this section we give a number of interesting theorems pertaining to LCs and their learnability. Most importantly, we characterize LLs (Theorem 4.6) and LCs (Theorem 4.7).

We start by observing an isomorphism between the hypothesis space and the class of learned languages. This gives a very important intuition about what kind of structures are learnable as LCs.

Theorem 4.1. *Let (V, f) be an LS. Then $(V, \sqsubseteq)$ and $(\mathcal{L}_f, \subseteq)$ are order-isomorphic, with order-isomorphism $L_f(\cdot)$.*

Proof. Clearly, $L_f(\cdot)$ is surjective. Regarding injectivity, let $a, b \in V$ with $L_f(a) = L_f(b)$. Let $D_a, D_b \subseteq \Sigma^*$ be finite sets such that $\bigsqcup_{x\in D_a} f(x) = a$ and $\bigsqcup_{x\in D_b} f(x) = b$. Clearly, $D_a, D_b \subseteq L_f(a) = L_f(b)$. Therefore, for all $x \in D_a \cup D_b, f(x) \sqcup a = a$ and $f(x) \sqcup b = b$, i.e., both a and b are upper bounds on the set $E = \{f(x) \mid x \in D_a \cup D_b\}$. As both a and b are the *least* upper bounds already on subsets of E, they both must be the least upper bound of E. The least upper bound of a set is unique, thus $a = b$.

Let $a, b \in V$ such that $a \sqsubseteq b$. Then we have

$$L_f(a) = \{x \in \Sigma^* \mid f(x) \sqsubseteq a\} \subseteq \{x \in \Sigma^* \mid f(x) \sqsubseteq b\} = L_f(b). \tag{2}$$

Let $a, b \in V$ such that $L_f(a) \subseteq L_f(b)$. Then we have

$$\{x \in \Sigma^* \mid f(x) \sqsubseteq a\} = L_f(a) \subseteq L_f(b) = \{x \in \Sigma^* \mid f(x) \sqsubseteq b\}. \tag{3}$$

Let $D \subseteq \Sigma^*$ be a finite set such that $\bigsqcup_{x\in D} f(x) = a$. Clearly, $D \subseteq L_f(a)$, and, thus, $D \subseteq L_f(b)$. Therefore, b is an upper bound on $\{f(x) \mid x \in D\}$. As a is the *least* upper bound of this set, we get $a \sqcup b = b$; thus, $a \sqsubseteq b$. □

This order isomorphism has the following important consequence. Given any set of languages $\mathcal{L}$ learnable as an LC, there is, up to isomorphism, only one LS to learn $\mathcal{L}$ with. We will use this fact occasionally when talking about an LC $\mathcal{L}$ without explicitly defining the associated LS.

From Theorem 4.1 we immediately get the following corollary.

Corollary 4.2. *Let (V, f) and (U, g) be two LSes with $\mathcal{L}_f \subseteq \mathcal{L}_g$. Then there is an order embedding $h : V \to U$.*

Proof. Define h such that

$$\forall v \in V : L_f(v) = L_g(h(v)). \tag{4}$$

Such a function exists, as $\mathcal{L}_f \subseteq \mathcal{L}_g$. We have, for all $a, b \in V$,

$$a \sqsubseteq_V b \Leftrightarrow L_f(a) \subseteq L_f(b) \Leftrightarrow L_g(h(a)) \subseteq L_g(h(b)) \Leftrightarrow h(a) \sqsubseteq_U h(b). \quad \square \tag{5}$$

We give the following proposition as three observations

Proposition 4.3. *We have the following.*

(i) *Let (V, f) be an LS and U a semi-lattice. Let $g : V \to U$ be an order embedding. Then $\mathcal{L}_f = \mathcal{L}_{g\circ f}$.*[5]
(ii) *Let $h : \Sigma^* \to \Sigma^*$ and let (V, f) be an LS. Then $h(\mathcal{L}_f) = \mathcal{L}_f$ iff, for all $x, y \in \Sigma^*, f(x) \sqsubseteq f(y) \Leftrightarrow f(h(x)) \sqsubseteq f(h(y))$.*
(iii) *Let $h : \Sigma^* \to \Sigma^*$ and $\mathcal{L}$ an LC. Then $h(\mathcal{L})$ is an LC.*

[5] Note that the semi-lattice associated with $g \circ f$ is the sub-ordering of V on range(g).

Proof. (i) We have, for all $v \in V$ (using the monotonicity of g for the second step),

$$\begin{aligned} L_f(v) &= \{x \in \Sigma^* \mid f(x) \sqsubseteq v\} & (6)\\ &= \{x \in \Sigma^* \mid g(f(x)) \sqsubseteq g(v)\} & (7)\\ &= L_{g\circ f}(g(v)). & (8)\end{aligned}$$

(ii) Regarding "$\Leftarrow$": From the assumptions we know, for all x, y with $f(x) = f(y)$, $f(h(x)) = f(h(y))$. Thus, there is $g : V \to V$ such that $\forall x \in \Sigma^* : g(f(x)) = f(h(x))$. From the assumptions, we get that g is order-embedding, and the result follows from (i).
Regarding "$\Rightarrow$": Let $x, y \in \Sigma^*$. Suppose $f(x) \sqsubseteq f(y)$. Thus, for all $L \in \mathcal{L}_f$, $y \in L \Rightarrow x \in L$. Hence, for all $L \in h(\mathcal{L}_f) = \mathcal{L}_f$, $h(y) \in L \Rightarrow h(x) \in L$. Thus, $f(h(x)) \sqsubseteq f(h(y))$. The opposite direction is analogous.

(iii) Let (V, f) be such that $\mathcal{L}_f = \mathcal{L}$. Let W be all elements from range$(f \circ h)$, closed under (finite) suprema. Then $(W, f \circ h)$ is an LC with $\mathcal{L}_{f\circ h} = h(\mathcal{L})$. □

After these first observations on the structure of LCs, we will now turn to properties of individual LCs. We start with a lemma which provides useful formulas for later theorems.

Lemma 4.4. *Let (V, f) be an LS. We have the following.*

(i) *For all $D \subseteq V$ such that $\bigsqcup D$ exists, $\bigcup_{v\in D} L(v) \subseteq L(\bigsqcup D)$.*
(ii) *For all $a \in V$, $L(a) = \Sigma^*$ iff $a = \top_V$.*
(iii) *If $\mathcal{L}_f$ is closed under (finite) union, then we have, for all $a, b \in V$, $L(a) \cup L(b) = L(a \sqcup b)$.*
(iv) *If $\mathcal{L}_f$ is closed under infinite union, then we have, for all $D \subseteq V$, $\bigsqcup D$ exists and $\bigcup_{v\in V} L(v) = L(\bigsqcup D)$.*
(v) *For all $D \subseteq V$ such that $\bigsqcap D$ exists, $\bigcap_{v\in D} L(v) = L(\bigsqcap D)$.*

Proof. (i) Let $D \subseteq V$, let $x \in \bigcup_{v\in D} L(v)$. Then there is a $v \in D$ such that $f(x) \sqsubseteq v$; thus, $f(x) \sqsubseteq \bigsqcup D$. Therefore, $x \in L(\bigsqcup D)$.
(ii) Let $a \in V$ be such that $L(a) = \Sigma^*$. Let $v \in V$ be such that $a \sqsubseteq v$, and let $D \subseteq \Sigma^*$ be finite such that $\bigsqcup_{x\in D} f(x) = v$. Then, as $L(a) = \Sigma^*$, for all $x \in D$, $f(x) \sqsubseteq a$. Thus, $v = \bigsqcup_{x\in D} f(x) \sqsubseteq a$. This shows $v = a$, and, therefore, $a = \top_V$. The converse is trivial.
(iii) Let $a, b \in V$. Let $L \in \mathcal{L}$ be the supremum of $L(a)$ and $L(b)$ with respect to $(\mathcal{L}, \subseteq)$ (i.e., the smallest language containing $L(a)$ and $L(b)$). As $L(a) \cup L(b) \in \mathcal{L}$, we have $L = L(a) \cup L(b)$. By Theorem 4.1, $(\mathcal{L}, \subseteq)$ and $(V, \sqsubseteq)$ are isomorphic with order isomorphism $L(\cdot)$. Thus $L(a \sqcup b)$ equals the supremum of $L(a)$ and $L(b)$ in $(\mathcal{L}, \subseteq)$, that is, $L(a \sqcup b) = L(a) \cup L(b)$.
(iv) Let $D \subseteq V$. Let $L \in \mathcal{L}$ be the supremum of all $L(v)$ for $v \in D$ with respect to $(\mathcal{L}, \subseteq)$ (i.e., the smallest language containing all $L(v)$). As $\bigcup_{v\in V} L(v) \in \mathcal{L}$ by closure under infinite union, we have $L = \bigcup_{v\in V} L(v)$. By Theorem 4.1, $(\mathcal{L}, \subseteq)$ and $(V, \sqsubseteq)$ are isomorphic with order isomorphism $L(\cdot)$. Thus $L(\bigsqcup_{v\in D} v)$ equals the supremum of all $L(v)$, $v \in D$, in $(\mathcal{L}, \subseteq)$, that is, $\bigcap_{v\in D} L(v) = L(\bigsqcap D)$.
(v) Let $D \subseteq V$ such that $\bigsqcap D$ exists. We have, for all $x \in \Sigma^*$,

$$x \in L(\bigsqcap D) \Leftrightarrow f(x) \sqsubseteq \bigsqcap D \Leftrightarrow \forall v \in D : f(x) \sqsubseteq v \Leftrightarrow x \in \bigcap_{v\in D} L(v). \quad \square$$

Now we get to one of the main theorems of this section. This theorem makes two statements on the quality of the hypothesis space of lattice learning and characterizes several closure properties of individual LCs in terms of their defining LSs. For this, we need the following notions of lattice theory.

Let a lattice V be given. We say that V is $\sqcap$-*complete* iff, for all (possibly infinite) sets $A \subseteq V$, A has an infimum. The definition of $\sqcup$-*completeness* is analogous. An element $v \in V$ is called $\sqcup$-irreducible iff, for all finite sets D, if $\bigsqcup D = v$ then $v \in D$.[6] V is called *distributive* iff, for all $u, v, w \in V$, $u \sqcup w = v \sqcup w$ and $u \sqcap w = v \sqcap w$ imply that $u = v$.

Theorem 4.5. *Let (V, f) be an LS. We have the following.*

(i) *$\lambda v, x.x \in L(v)$ is computable (i.e., $(L(v))_{v\in V}$ is uniformly decidable).*
(ii) *If (V, f) is polytime, then $\lambda v, x.x \in L(v)$ is computable in polynomial time (i.e., $(L(v))_{v\in V}$ is uniformly decidable in polynomial time).*
(iii) *$\mathcal{L}_f$ is closed under intersection iff V is a lattice.*
(iv) *$\mathcal{L}_f$ is closed under infinite intersection iff V is a $\sqcap$-complete lattice.*
(v) *$\mathcal{L}_f$ is closed under finite union iff the following holds. Each $v \in$ range(f) is $\sqcup$-irreducible and, for all $u, v, w \in V$ such that w is incomparable with both u and v, and $u \sqcup w = v \sqcup w$, we have $L(u) \setminus L(w) = L(v) \setminus L(w)$.*
(vi) *$\mathcal{L}_f$ is closed under infinite union iff $\mathcal{L}_f$ is closed under union, V is $\sqcup$-complete and V does not have any infinitely ascending chains.*
(vii) *Let V be a lattice and $\mathcal{L}_f$ be closed under finite union. Then V is distributive.*[7]

[6] This notion is especially interesting in the present context, since any x such that $f(x)$ is $\sqcup$-irreducible intuitively provides information about the language $L(f(x))$ that cannot be obtained from any number of elements $u \in L(f(x))$ with $f(u) \neq v$.

[7] Note that there are LSs on distributive lattices V where all elements from range(f) are $\sqcup$-irreducible such that the associated LCs are *not* closed under finite union. Consider the following set of regular languages $\{\emptyset, L((a^2)^*), L((b^2)^*), L(a^*), L(b^*), L((a^2)^*|(b^2)^*), \Sigma^*\}$. This set of languages is not closed under union (for example, $L(a^*|(b^2)^*)$ is missing), but the lattice structure is distributive and all elements from range(f) are $\sqcup$-irreducible.

(viii) *Suppose V is a lattice. $\mathcal{L}_f$ is closed under complements iff $\mathcal{L}$ is closed under union, V is boolean and $\bot_V \notin \text{range}(f)$.*
(ix) *Let (V, f) be an LS. Then $\mathcal{L}_f$ is closed under reversal iff, for all $x, y \in \Sigma^*$, $f(x) \sqsubseteq f(y) \Leftrightarrow f(\text{reverse}(x)) \sqsubseteq f(\text{reverse}(y))$.*

Proof. (i) We have $\lambda v, x.[x \in L(v)] = \lambda v, x.[\phi_f^{it}(v, x) = v]$ by consistency and conservativeness of ϕ_f. Clearly, $\lambda v, x.[\phi_f^{it}(v, x) = v]$ is computable.
(ii) Using Theorem 2.9(ii), analogous to the proof of (i) just above.
(iii) "$\Rightarrow$": Follows from the isomorphie given in Theorem 4.1. "$\Leftarrow$": Follows directly from Lemma 4.4(v).
(iv) Same as (iii).
(v) "$\Rightarrow$": Suppose $\mathcal{L}$ is closed under union. Let $v \in \text{range}(f)$. By way of contradiction, suppose v is *not* $\sqcup$-irreducible; thus, there are $u, v, w \in V$ with $u \neq v \neq w$ such that $u \sqcup w = v$. Let $x \in \Sigma^*$ is such that $f(x) = v$. Then $x \in L(v) \setminus (L(u) \cup L(w))$, but we have $L(u) \cup L(w) \underset{\text{Lemma 4.4(iii)}}{=} L(u \sqcup w) = L(v)$, a contradiction.

Now suppose there are $u, v, w \in V$ such that w is incomparable with both u and v and $u \sqcup w = v \sqcup w$. Suppose, by way of contradiction, $L(u) \setminus L(w) \neq L(v) \setminus L(w)$. Without loss of generality, suppose $(L(u) \setminus L(w)) \setminus (L(v) \setminus L(w)) \neq \emptyset$. Let $x \in L(u) \setminus (L(v) \cup L(w))$. We have $f(x) \sqsubseteq u \sqcup w = v \sqcup w$. Thus, $x \in L(v \sqcup w) \underset{4.4\text{(iii)}}{=} L(v) \cup L(w)$, a contradiction.

"$\Leftarrow$": Let $u, w \in V$. We show $L(u) \cup L(w) = L(u \sqcup w)$. Suppose, by way of contradiction, there exists $x \in L(u \sqcup w) \setminus (L(u) \cup L(w))$. Thus, u and w are incomparable. As $x \notin L(u)$, $f(x) \not\sqsubseteq u$, similarly for w.

First, suppose $f(x)$ is comparable with both u and v. Thus, $f(x)$ is an upper bound to both u and w; hence, $f(x) = u \sqcup w$. This contradicts $f(x) \in \text{range}(f)$ being $\sqcup$-irreducible.

Second, suppose $f(x)$ is not comparable with one of u and w (without loss of generality, not comparable with w). Let v be such that

$$v = \begin{cases} f(x) & \text{if } u \sqcup w = f(x) \sqcup w; \\ f(x) \sqcup u, & \text{otherwise.} \end{cases}$$

Then, as $f(x) \sqsubseteq u \sqcup w$, $u \sqcup w = v \sqcup w$. Furthermore, w is incomparable with both u and v. Thus, we get $L(u) \setminus L(w) = L(v) \setminus L(w)$, but the right set contains x, while the left does not, a contradiction.
(vi) "$\Rightarrow$": Trivially, $\mathcal{L}_f$ is closed under finite union. $\sqcup$-completenes follows from Lemma 4.4(iv). If V contained an infinitely ascending chain, then $\mathcal{L}_f$ contained an infinitely ascending chain and its supremum, a direct contradiction to an early language learning result by Gold [20] (he showed that no class of languages containing all finite sets as well as an infinite set can be **TxtEx**-learned).

"$\Leftarrow$": Let $D \subseteq V$. Let $v \in D$. Recursively define a sequence $(v_i)_i$ by $v_0 = v$ and, for all i, we let v_{i+1} be any element from D such that $v_{i+1} \not\sqsubseteq \bigsqcup_{j \leq i} v_j$, or v_i, if no such element exists. As V has no infinitely ascending chains, $(\bigsqcup_{j \leq i} v_j)_i$ is not infinitely ascending and is almost everywhere equal some u. Thus, for all $w \in D$, $w \sqsubseteq u$. Thus, using that u is the supremum of finitely many elements from D, u is the supremum of all of D, i.e., $u = \bigsqcup D$. Let i be such that $\bigsqcup_{j \leq i} v_j = u$. As $\mathcal{L}_f$ is closed under finite union, we have

$$\bigcup_{j \leq i} L(v_j) \underset{4.4\text{(iii)}}{=} L\left(\bigsqcup_{j \leq i} v_j\right) = L(u) = L\left(\bigsqcup D\right) \underset{4.4\text{(i)}}{\supseteq} \bigcup_{v \in D} L(v) \supseteq \bigcup_{j \leq i} L(v_j).$$

Thus, $L(\bigsqcup D) = \bigcup_{v \in D} L(v)$ as desired.
(vii) Let $u, v, w \in V$ with $u \sqcap w = v \sqcap w$ and $u \sqcup w = v \sqcup w$. The conclusion is straightforward if w is comparable with either u or v. Thus, suppose that w is not comparable with both u and v. We get $L(u) \setminus L(w) = L(v) \setminus L(w)$ from (v). Using (iii) and Lemma 4.4(v), we get

$$\begin{aligned} L(u) &= (L(u) \setminus L(w)) \cup (L(u) \cap L(w)) \\ &= (L(v) \setminus L(w)) \cup (L(u \sqcap w)) \\ &= (L(v) \setminus L(w)) \cup (L(v \sqcap w)) \\ &= (L(v) \setminus L(w)) \cup (L(v) \cap L(w)) \\ &= L(v). \end{aligned}$$

Thus, $u = v$.
(viii) "$\Rightarrow$": Suppose $\mathcal{L}$ is closed under complements. By DeMorgan's laws and (iii), $\mathcal{L}$ is closed under union. Let v be a grammar for $\overline{L(\bot_V)}$. Then $\bot_V \sqsubseteq v$, therefore $L(\bot_V) \subseteq L(v) = \overline{L(\bot_V)}$. Thus, $L(\bot_V) = \emptyset$ and, hence, $\bot_V \notin \text{range}(f)$. Further, $L(v) = \Sigma^*$, hence, by Lemma 4.4(ii), $v = \top_V$.

Let $\bar{\cdot}$ be such that, for all $a \in V$, $\bar{a}$ is a grammar for $\overline{L(a)}$. Let $a \in V$. For all $x \in \Sigma^*$, we have $x \in L(a)$ or $x \in \overline{L(a)} = L(\bar{a})$. Thus, $f(x) \sqsubseteq a \sqcup \bar{a}$. Therefore, $\Sigma^* = L(a \sqcup \bar{a})$. From Lemma 4.4(ii) we get $a \sqcup \bar{a} = \top_V$.

"$\Leftarrow$": For each $a \in V$ and $x \in \Sigma^*$, we have

$$x \in L(a) \cap L(\bar{a}) \Leftrightarrow f(x) \sqsubseteq a \sqcap \bar{a} \Leftrightarrow f(x) = \bot_V \Leftrightarrow x \in \emptyset. \tag{9}$$

Thus, $L(a) \cap L(\overline{a}) = \emptyset$. As $\mathcal{L}$ is closed under union, let v be a grammar for $L(a) \cup L(\overline{a})$. We have

$$L(v) = L(a) \cup L(\overline{a}) = L(a \sqcup \overline{a}) = L(\top_V). \tag{10}$$

Now we have $v = \top_V$ and, using Lemma 4.4(i), $L(a) \cup L(\overline{a}) = L(v) = \Sigma^*$. Thus, $L(\overline{a}) = \overline{L(a)}$.

(ix) This follows directly from Proposition 4.3(ii). □

Now we get to the second main theorem of this section, which shows that all learners having a certain subset of the properties listed above in Theorem 2.9 can necessarily be expressed as LLs.

Theorem 4.6. *Let $h \in \mathcal{R}$. The following are equivalent.*

(i) *There is an LS (V, f) such that $h = \phi_f$.*
(ii) *h **TxtEx**-learns $\mathcal{L}$ set-drivenly, globally consistently, locally conservatively and strongly monotonically.*
(iii) *There is a 1–1 $L(\cdot)$ and a computable function t such that, for all x, v, $t(x, v)$ halts iff $x \in L(v)$ and, for all $\sigma \in \mathbb{S}\mathrm{eq}$, $L(h(\sigma))$ is the $\subseteq$-minimum element of **TxtEx**(h) containing all of* content(σ).

Proof. We have that (i) implies (iii) by basic properties of the LLs (see Theorem 2.9).

Regarding (iii) implies (ii): set-drivenness, global consistency and local conservativeness are straightforward. Then h is prudent [11, Proposition 21]. Concerning strong monotonicity we have the following. Let $D, D' \subseteq \Sigma^*$ with $D \subseteq D'$. Then $D \subseteq L(h^{set}(D'))$, while $L(h^{set}(D))$ is the $\subseteq$-minimum element of **TxtEx**(h) containing all of D (from prudence we have $L(h^{set}(D)) \in$ **TxtEx**(h)). Hence, $L(h^{set}(D)) \subseteq L(h^{set}(D'))$.

Regarding (ii) implies (i): As h learns set-drivenly, let h^{set} be such that, for all sequences σ, $h(\sigma) = h^{set}(\mathrm{content}(\sigma))$. Note that, for all D, D' such that $h^{set}(D) = h^{set}(D')$, we have

$$h^{set}(D) = h^{set}(D \cup D') \tag{11}$$

by consistency and conservativeness.

Let $V = \mathrm{range}(h^{set})$ and define $\sqcup$ by

$$\forall D_0, D_1 : h^{set}(D_0) \sqcup h^{set}(D_1) = h^{set}(D_0 \cup D_1). \tag{12}$$

To show $\sqcup$ to be well-defined: Let D_0, D_0', D_1, D_1' be such that $h^{set}(D_0) = h^{set}(D_0')$ and $h^{set}(D_1) = h^{set}(D_1')$. We have

$$\begin{aligned} & D_0 \cup D_0' \cup D_1 \cup D_1' & (13)\\ & \underset{\text{consistency}}{\subseteq} L(h^{set}(D_0)) \cup L(h^{set}(D_0')) \cup L(h^{set}(D_1)) \cup L(h^{set}(D_1')) & (14)\\ & = L(h^{set}(D_0)) \cup L(h^{set}(D_1)) & (15)\\ & \underset{\text{strict monotonicity}}{\subseteq} L(h^{set}(D_0 \cup D_1)). & (16) \end{aligned}$$

Similarly, we get

$$D_0 \cup D_0' \cup D_1 \cup D_1' \subseteq L(h^{set}(D_0' \cup D_1')). \tag{17}$$

From conservativeness we get

$$h^{set}(D_0 \cup D_1) = h^{set}(D_0 \cup D_0' \cup D_1 \cup D_1') = h^{set}(D_0' \cup D_1'). \tag{18}$$

This shows $\sqcup$ to be well-defined.

We define $\sqsubseteq$ by, for all $a, b \in V$, $a \sqsubseteq b$ iff $a \sqcup b = b$. It is easy to verify that $\sqsubseteq$ is a partial order on V (see [37, Theorem 2.1]).

Let $f : \Sigma^* \to V, x \mapsto h^{set}(\{x\})$. Then, for all $\sigma \in \mathbb{S}\mathrm{eq}$, $\phi_f(\sigma) = h(\sigma)$.

Obviously, f and suprema in V are computable. Furthermore, for each $v \in V$ there is a finite set $D \subseteq \mathrm{range}(f)$ such that $\bigsqcup_{x \in D} x = v$.

This shows that (V, f) is an LS as desired. □

Theorem 4.6 allows for a straightforward corollary on the level of LCs. The next theorem gives this corollary and one more characterization of LCs.

Theorem 4.7. *Let $\mathcal{L}$ be a set of languages. The following are equivalent.*

(i) *$\mathcal{L}$ is an LC.*
(ii) *$\mathcal{L}$ can be **TxtEx**-learned by a globally consistent, locally conservative, set-driven and strongly monotonic learner.*
(iii) *There is a 1–1 $L(\cdot)$ such that there is a computable function t such that, for all x, v, $t(x, v)$ halts iff $x \in L(v)$ and a (total) computable function g such that, for all $D \subseteq \Sigma^*$ $L(g(D))$ is the $\subseteq$-minimum element of $\mathcal{L}$ containing D.*
(iv) *$\mathcal{L}$ can be **TxtEx**-learned by a strongly monotonic set-driven learner using a uniformly decidable hypothesis space.*

Proof. We have that (i), (ii) and (iii) are equivalent by Theorem 4.6.

Further, (i) implies (iv) by Theorems 2.9 and 4.5.

Regarding (iv) implies (ii) we have the following. Suppose $h \in \mathcal{P}$ **TxtEx**-learns $\mathcal{L}$ strongly monotonically and set-drivenly using a uniformly decidable hypothesis space L. We will define a learner $h' \in \mathcal{P}$ using hypotheses in a uniformly decidable hypothesis space L'-system defined such that each hypothesis is a pair of a natural number and a finite sequence; in particular, for all $e, \sigma, L'(e, \sigma) = L(e) \cup \mathrm{content}(\sigma)$. We define a learner h' as follows.[8]

$$\forall \sigma \in \mathbb{S}\mathrm{eq} : h'(\sigma) = \begin{cases} h'(\sigma^-), & \text{if } \sigma \neq \emptyset \ \wedge \ \mathrm{content}(\sigma) \subseteq L'(h'(\sigma^-)); \\ (h(\sigma), \sigma), & \text{else if } \mathrm{content}(\sigma) \not\subseteq L(h(\sigma)); \\ (h(\sigma), \emptyset), & \text{otherwise.} \end{cases} \tag{19}$$

It is easy to see that h' is set-driven, globally consistent, locally conservative and strongly monotonic. Furthermore, h' **TxtEx**-learns $\mathcal{L}$ as desired. □

We end this section with another sufficient condition for a language to be an LC.

Proposition 4.8. *Let $\mathcal{L}$ be a class of languages closed under intersection and* **TxtEx***-learnable set-drivenly, globally consistently and locally conservatively as witnessed by $h \in \mathcal{P}$. Then h is strongly monotone, and, in particular, $\mathcal{L}$ is an LC.*

Proof. Let $\sigma \in \mathbb{S}\mathrm{eq}, x \in \Sigma^*$. We need to show $L(h(\sigma)) \subseteq L(h(\sigma \diamond x))$. Let $L_0 = L(h(\sigma)) \cap L(h(\sigma \diamond x))$. As h is globally consistent, we have $\mathrm{content}(\sigma) \subseteq L_0$. Note that $L_0 \in \mathcal{L}$, as $\mathcal{L}$ is closed under intersection. Let $T \supseteq \sigma$ be a text for L_0. As h is globally conservative and **TxtEx**-learns L_0, we have, for all $k \geq \mathrm{len}(\sigma)$, $h(T[k]) = h(\sigma)$. Thus, $L(h(\sigma)) = L_0$. We now have

$$L(h(\sigma)) = L_0 = L(h(\sigma)) \cap L(h(\sigma \diamond x)) \subseteq L(h(\sigma \diamond x)). \tag{20}$$

We now get that $\mathcal{L}$ is an LC by Theorem 4.7. □

5. Complex examples of lattice classes

In this section, we provide additional examples of lattice classes that are more complex than the ones given earlier. In particular, most of these lattices are infinite. We begin with the pattern languages [2], and then discuss monomials. We conclude by showing that the function-distinguishable language classes [16] are LCs.

Definition 5.1. Let Σ be an alphabet and let X be a countably infinite set (of *variables*) disjoint from Σ.

Let $\mathbb{P}\mathrm{at} = (\Sigma \cup X)^*$ be the set of all patterns. For any $\pi \in \mathbb{P}\mathrm{at}$, with $w_0, \ldots w_{n+1} \in \Sigma^*$ and $x_0, \ldots x_n \in X$ such that $\pi = w_0 x_0 w_1 x_1 \ldots x_n w_{n+1}$, let

$$L(\pi) = \{w_0 v_{x_0} w_1 v_{x_1} \ldots v_{x_n} w_{n+1} \mid \forall x \in X : v_x \in \Sigma^* \setminus \{\varepsilon\}\}$$

denote the set of all strings *matching* the pattern π. We call any L such that there is a pattern π with $L = L(\pi)$, a (non-erasing) *pattern language*. For each $w \in \Sigma^*$, let $\mathrm{pat}(w) = \{\pi \in \mathbb{P}\mathrm{at} \mid w \in L(\pi)\}$ denote the set of patterns matched by w. Note that, for each $w \in \Sigma^*$, $\mathrm{pat}(w)$ is finite.

The pattern languages are not learnable globally consistently and iteratively in a *non-redundant* hypothesis space, see [10, Corollary 12]. The usual iterative algorithm was first published in [32].

Theorem 5.2. *This theorem follows [33]. For any finite set $D \subseteq \Sigma^*$, we let $\mathrm{pat}(D) = \bigcap_{w \in D} \mathrm{pat}(w)$.*[9] *Let V_{pat} be the lattice $\{\mathrm{pat}(D) \mid D \subseteq \mathrm{Pow}_{\mathrm{fin}}(\Sigma^*)\}$ with order relation $\supseteq$.*[10] *Then (V_{pat}, pat) is an LS.*

Now ϕ_{pat} learns the pattern languages globally consistently and iteratively (as well as with all other properties as given in Theorem 2.9). Note that some of the grammars of (V_{pat}, pat) are not for pattern languages, for example $\mathrm{pat}(\{a^3, b^4\}) = \{x_1, x_1x_2, x_1x_2x_3, x_1x_1x_2, x_1x_2x_1, x_1x_2x_2\}$.[11]

Also note: One can code the elements of V_{pat}, as all but $\perp_{V_{pat}}$ are finite sets.

We can generalize the construction given for the pattern languages.

Theorem 5.3. *Let $(L_i)_i$ be an enumeration of languages such that there is a computable function h such that, for all $x \in \Sigma^*$, $h(x)$ is the finite set of all i with $x \in L_i$.*[12] *Then a superset of $\{L_i \mid i \in \mathbb{N}\}$ is an LC.*

8 For any non-empty sequence σ, we let σ^- denote the sequence derived from σ by deleting the last element.

9 By convention, we let $\mathrm{pat}(\emptyset) = \mathbb{P}\mathrm{at}$.

10 Note that the order is inverted with respect to the usual powerset lattice.

11 Note that the pattern languages are not closed under intersection [29]. In particular, the language generated by the grammar $\{x_1, x_1x_2, x_1x_2x_3, x_1x_1x_2, x_1x_2x_1, x_1x_2x_2\}$ is the intersection of the pattern languages recognized by $x_1x_1x_2$, $x_1x_2x_1$ and $x_1x_2x_2$. In this intersection are no words of length 2, and the only words of length 3 are clearly *aaa* and *bbb*. Hence, the only pattern that might describe the intersection is $x_1x_1x_1$, which does not include b^4 (which is in the intersection).

12 The property of having only finitely many possible conjectures including a given datum is called *finite thickness*. Finite thickness is a sufficient condition for **TxtEx**-learnability [2].

Proof. We let V be the range of h and close it under intersection. Then $(V, \cap)$ is a semi-lattice, and (V, h) is an LS. □

Note that this construction will not yield grammars for languages outside of $(L_i)_i$ if $(L_i)_i$ is closed under intersection. This kind of closure of a class under finite intersection was already studied in [31].

Our next example concerns *monomials* over a set of *variables* $\{x_1, \ldots, x_n\}$. Intuitively, a monomial is a boolean function defined as the conjunction of any number of literals (a literal is a variable or its negation). We are interested in the set of all input tuples on which the monomial will evaluate to "true". We represent each monomial for which there are such tuples by a tuple in turn: If, for some $k \leq n$, variable x_k appears in the conjunctive definition of the monomial we put a 1, if the negation appears a 0 and ? otherwise. Now a monomial will evaluate to true on all and only the tuples which match the monomial tuple on all non-? places. The following definition makes these ideas formal.

Definition 5.4. Let $n \in \mathbb{N}$. A *monomial of n variables* is an element from $\{0, 1, ?\}^n$.

Let $\sqsubseteq$ be the order on monomials of n variables such that

$$\forall x, y \in \{0, 1, ?\}^n : x \sqsubseteq y \Leftrightarrow (\forall k < n : y(k) \neq ? \Rightarrow x(k) = y(k)).$$

Let V_{mon-n} be the (finite) lattice on $\{\bot\} \cup \{0, 1, ?\}^n$ induced by letting $\bot$ be the minimum element and all other relations be as given by $\sqsubseteq$. This lattice has a maximum element $?^n$ and the set of atoms $\{0, 1\}^n$.

For any monomial m, let

$$L(m) = \{x \in \{0, 1\}^n \mid x \sqsubseteq m\}$$

denote the set of all strings *matching* the monomial m.

Theorem 5.5. *Let $n \in \mathbb{N}$, let id be the identity on $\{0, 1\}^n$. Then (V_{mon-n}, id) is an LS. In particular, ϕ_{id} learns the class of all monomial languages.*

Fernau [16] introduced the notion of *function distinguishable languages* (FDLs). We define these in automata-theoretic terms. A finite-state automaton $\mathcal{A} = (Q, \Sigma, q_0, Q_F, \delta)$ is a tuple where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the start state, Q_F are the final states and δ is the transition function with domain $Q \times \Sigma$ and co-domain the powerset of Q. The transition function is extended recursively in the usual way so its domain is $Q \times \Sigma^*$.

A function f is *distinguishing* iff its domain is Σ^*, its codomain is a finite set, and if $f(w) = f(z)$ implies $f(wu) = f(zu)$ for all $u, w, z \in \Sigma^*$. Consider any finite-state automaton $\mathcal{A} = (Q, \Sigma, q_0, Q_F, \delta)$ and distinguishing function f. For all $q \in Q$ and for all $x \in \Sigma^*$ such that $\delta(q_0, x) = q$, define $\hat{f} : Q \to range(f)$ to be $\hat{f}(q) = f(x)$. Then $\mathcal{A}$ is a *f-distinguishable* automaton iff

(i) $\mathcal{A}$ is deterministic (so $\forall q \in Q, a \in \Sigma$, it is the case that $|\delta(q, a)| \leq 1$), and
(ii) for all distinct states $q_1, q_2 \in Q$, if q_1 and q_2 are both final or if there is $a \in \Sigma$ such that $\delta(q_1, a) = \delta(q_2, a)$ then $\hat{f}(q_1) \neq \hat{f}(q_2)$.

A language is f-distinguishable if and only if there is a f-distinguishable automaton accepting it. The class of f-distinguishable languages is denoted $\mathcal{L}_{f\mathrm{DL}}$.

The following shows that the concept of FDLs is subsumed by the concept of LCs, while the concept of LCs is *not* subsumed by the concept of FDLs.

Theorem 5.6.

$$\left\{\mathcal{L}_{f\mathrm{DL}} \mid f \text{ is distinguishable}\right\} \subset \left\{\mathcal{L}_{(V,f)} \mid (V, f) \text{ is a lattice space}\right\}.$$

The inequality is witnessed by a class of regular languages as stated below.

Proof. "$\neq$": We argue that the class of all finite languages $\mathcal{L}_{fin}$ is an LC but not function-distinguishable. To see that it is a lattice class, consider the lattice of all finite sets with inclusion as the order, and the function f is given by the mapping of all $w \in \Sigma^*$ to $\{w\}$. On the other hand, each $\mathcal{L}_{f\mathrm{DL}}$ is *not* closed under union (see [16, Property 17]), but, trivially, $\mathcal{L}_{fin}$ is.

"$\subseteq$": Consider any $\mathcal{L}_{f\mathrm{DL}}$. Let h be the learner for $\mathcal{L}_{f\mathrm{DL}}$ given in [16, § 6]. By Fernau [16, Theorem 35], h fulfills the condition of Theorem 4.6(iii). Hence, h is a Lattice Learner by Theorem 4.6 and $\mathcal{L}_{f\mathrm{DL}}$ is an LC. □

For the reader familiar with [16] we specify a concrete LS (V, f) such that ϕ_f learns the class of f'-DLs for any distinguishing function $f' : \Sigma^* \to X$.

Define V as the set of all stripped[13] f'-distinguishable DFA $\cup \{(\{q_0\}, \Sigma, q_0, \emptyset, \emptyset)\}$, and $\sqsubseteq$ such that $B_1 \sqsubseteq B_2$ iff $L(B_1) \subseteq L(B_2)$ for $B_1, B_2 \in V$.

Obviously, V is a partially ordered set. $(V, \sqsubseteq)$ is also an upper semi-lattice – the supremum B of B_1, B_2 is obtained as follows: Compute the stripped minimal DFA B_0 for $L(B_1) \cup L(B_2)$ (algorithms can be found in the literature). If $B_0 \in V$ then $B := B_0$. Else build a finite positive sample set I_+ by first adding all shortest strings leading to an accepting state

[13] An automaton is stripped when taking away any state or transition would change the language recognized by the automaton.

in B_0. Then, for every hitherto unrepresented transition $\delta(q_1, a) = q_2$ ($a \in \Sigma$) of B_0, add to I_+ the string obtained from concatenating a string leading to q_1 with a with a string leading from q_2 to an accepting state. Use the learner h from [16] on I_+. By Lemma 34 and Theorem 35 in [16] the result is a stripped DFA recognizing the smallest f'-distinguishable language containing $L(B_1) \cup L(B_2)$, and since the elements of V are all stripped there is only one such DFA in V, which is the supremum of B_1 and B_2. Also note that $(V, \sqsubseteq)$ has a minimum element $\bot_V = (\{q_0\}, \Sigma, q_0, \emptyset, \emptyset)$.

For any distinguishing function $f' : \Sigma^* \to X$ define $f : \Sigma^* \to V$ by setting $f(w) := A_w$ where A_w is the minimal stripped DFA with $L(A_w) = \{w\}$ (A_w is f'-distinguishable by Fernau [16], Lemma 15). We show that (V, f) is an LS.

Obviously, f is computable. For each $v \in V$ there is a finite set $D \subseteq \text{range}(f)$ such that $\bigsqcup_{x \in D} x = v$: Take any two elements $B_1, B_2 \in V$ such that $B_1 \sqcup B_2 = v$ and construct the set I_+ as specified above. We can set $D := I_+$.

Thus, the class of f'-DLs is learnable by ϕ_f.[14]

It follows that the famously-studied class of k-reversible languages [4], which are function distinguishable [16], can be learned by lattice learners.

6. Query learning of LCs

This section is concerned with learning LCs from queries [5]. We address the issue from a more Grammatical Inference-oriented view. For example, some concrete algorithms are given and complexity questions are considered.

Definition 6.1. Let (V, f) be an LS and $v \in V$ be the learning target.[15] A *membership query (MQ)* for $w \in \Sigma^*$ and $L \subseteq \Sigma^*$ is a query '$w \in L$?' receiving an answer from $\{0, 1\}$ with $\text{MQ}(w) = 1$ if $w \in L$ and $\text{MQ}(w) = 0$ otherwise.[16] An MQ-learner for an LS (V, f) is an algorithm that uses MQs instead of text to produce its sequence of hypotheses, outputting one hypothesis after each MQ.

An *equivalence query (EQ)* for $v_0 \in V$ is a query '$v_0 = v$?' receiving an answer from $\Sigma^* \cup \{\texttt{yes}\}$ ($\Sigma^* \cap \{\texttt{yes}\} = \emptyset$) such that $\text{EQ}(v_0) = \texttt{yes}$ for $L(v_0) = L(v)$ and $\text{EQ}(v_0) = c$ where c is in the symmetric difference of v_0 and v otherwise.

Let (V, f) be an LS. As $\mathcal{L}_f$ is **TxtEx**-learnable, $\mathcal{L}_f$ is also **TxtEx**-learnable in the limit from MQs: Consider a learner just querying all strings $w \in \Sigma^*$ in length-lexical order, keeping all w with $\text{MQ}(w) = 1$, adding a pause on all other w; this way, we build a text for the target, which we can feed to our LL.

If we are interested in complexity, unfortunately in general we cannot bound the number of MQs needed in any interesting way. Let (V, f) be an LS. Given an MQ-learner h for (V, f), we call the number of queries that h makes on any language $L \in \mathcal{L}_f$ before having converged to a grammar for L the *query complexity of h on L*. For each $v \in V$, define $\mathbb{T}_v := \{T \subseteq \text{range}(f) \mid \bigsqcup_{t \in T} t = v\}$ and let T_0 be an element of $\mathbb{T}_v$ with minimal cardinality. Obviously, $|T_0|$ is a lower bound on the query complexity. However, there are LCs with properties that allow more specific statements as follows. This theorem applies, for example, to the class of all k-factor languages or the class of all k-piecewise testable languages.

Theorem 6.2. *Let (V, f) be an LS such that V is finite and each $v \in V$ is the supremum of a set of atoms. Let m be the number of atoms of V. Then the worst case query complexity using only MQs is (exactly!) m.*

Proof. Clearly, querying one string x_v with $f(x_v) = v$ for each atom $v \in V$ and outputting the supremum of $\{x_v \mid \text{MQ}(x_v) = 1\}$ gives the desired MQ-learner.

We show that the worst case query complexity is lower bounded by m by an adversary argument. Suppose h is an MQ-learner for $\mathcal{L}_f$. The adversary answers the first $m - 1$ queries for w with $f(x) \neq \bot_V$ with 0 (and all with $f(x) = \bot_V$ with 1, as necessary). Let v be an atom v such that h has not queried an x with $f(x) = v$ within the first up to $m - 1$ queries. Suppose, by way of contradiction, h does not make more queries. Thus, h cannot learn both $L_f(v)$ and $L_f(\bot_V)$, a contradiction. Otherwise, if h's current hypothesis is $\bot_V$, the adversary will henceforth answer consistently with $L_f(v)$ and vice versa. Thus, h cannot have converged yet and will need another query before convergence.

Note that, in the case of Theorem 6.2, the MQ-learner is finite, i.e., using m queries and then outputting a single hypothesis afterwards. $\square$

We give another theorem regarding learnability with membership queries, this time about a specific infinite linear lattice.

Theorem 6.3. *Let (V, f) be an LS such that V equals $\mathbb{N}$ with the usual ordering. Suppose there is a computable $g \in \mathcal{R}$ such that $\forall n \in \mathbb{N} : f(g(n)) = n$ (i.e., we can compute, for any natural number, a string mapping to that number). Then the worst case query complexity to learn any L_n with $n \in \mathbb{N}$ is $O(\log n)$.*

[14] Note that in a concrete implementation we would not have to construct I_+ when computing suprema in V as we can just use the text seen so far. Also, it seems relatively easy to define an iterative version of the learner from [16].

[15] To be precise, the concept to infer is a language. However, as no two elements of V define the same language (see Footnote 4), our potential targets directly correspond to elements of V.

[16] Algorithmically, using MQs only makes sense if the membership problem is decidable. Note that, for an LS (V, f), a MQ for $w \in \Sigma^*$ amounts to checking if $f(w) \sqsubseteq v$.

Proof. We use a doubling algorithm with two phases as follows. Let learner h first query $g(0)$, then $g(1), g(2), g(4), g(8)$ and so on, until the first 0 as answer (on some 2^{n_0}). Thus, $\mathrm{MQ}(g(2^{n_0})) = 1$ and $\mathrm{MQ}(g(2^{n_0+1})) = 0$. Now h makes a binary search in the interval between 2^{n_0} and 2^{n_0+1}.

The total query complexity of h on any $n \in \mathbb{N}$ is $\lceil \log n \rceil$ for the first phase searching for the right interval. This search will end with some n_0 with $2^{n_0} \leq n$. Thus, the binary search in the second phase uses at most $\lceil \log(2^{n_0+1} - 2^{n_0}) \rceil \leq \lceil \log n \rceil$ queries. This gives the desired query complexity bound. □

Note that, on linear orderings, using MQs is equivalent to using EQs. Hence, Theorem 6.3 translates trivially to EQs. However, for nonlinear orderings, the situation changes if we allow EQs instead of MQs:

Theorem 6.4. *Let (V, f) be an LS. Then $\mathcal{L}_f$ is learnable from EQs. In particular, there is a learner h such that, for all $v \in V$, the maximum length of any ascending path from $\perp_V$ to v (if existent) is (exactly) the worst case EQ query complexity of h on $L_f(v)$.*

Proof. We design an EQ learner h as follows. Let D be the set of all EQ counterexamples received by h so far. Then we query $\bigsqcup_{x \in D} f(x)$. Clearly, h EQ-learns $\mathcal{L}_f$.

Let $v \in V$ and p be a maximal sequence of increasing elements of V with first element $\perp_V$ and last element v. We inductively show that there are answers to EQs of h such that the sequence of outputs of h is p. The base case of the induction is clear. Now let D and i be such that h got D as EQ counterexamples, $i = |D|$ and $\bigsqcup_{x' \in D} f(x') = p(i)$. Let $x \in L_f(p(i+1)) \setminus L_f(p(i))$ (this set is non-empty, see Theorem 4.1). Then $p(i) \sqsubset f(x) \sqcup \bigsqcup_{x' \in D} f(x') \sqsubseteq p(i+1)$. Thus, as p is *maximal* ascending path, $f(x) \sqcup \bigsqcup_{x' \in D} f(x') = p(i+1)$. Giving x as an EQ counterexample is thus as required.

This establishes the lower bound on the worst case query complexity, the upper bound is similar and straightforward. □

7. VC-dimension of LCs

One model of stochastic learnability is the PAC (probabilistically approximately correct) learning model [49]. Learnability in this model was shown to be strongly connected to the *VC-dimension* of the class in question [50,7]. In this section, we characterize the VC-dimension of lattice classes in terms of their lattice spaces in Theorem 7.1 and give an easy sufficient condition in Corollary 7.2. As an anonymous reviewer pointed out, a small VC-Dimension is not directly connected to *efficient* PAC-learnability. The analysis of efficiently PAC-learnable lattice classes is left as future work.

For any set $\mathcal{L} \subseteq \mathrm{Pow}(\Sigma^*)$, we denote with $\mathrm{VC}(\mathcal{L})$ the supremum of the cardinalities of all sets S such that $\{S \cap L \mid L \in \mathcal{L}\} = \mathrm{Pow}(S)$ (such a set is said to be *shattered by* $\mathcal{L}$).

We characterize the VC-dimension of an LC in the following theorem.

Theorem 7.1. *Let (V, f) be an LS. Let $\mathcal{M}$ be the set of all $M \subseteq \mathbb{N}$ such that $(\mathrm{Pow}(M), \subseteq)$ can be order-embedded into V. Then*

$$\sup_{M \in \mathcal{M}} \mathrm{card}(M) = \mathrm{VC}(\mathcal{L}_f).$$

Proof. Regarding "$\leq$": Consider any $M \in \mathcal{M}$ and $W \subseteq V$ such that there is an isomorphism h from $(\mathrm{Pow}(M), \subseteq)$ to $(W, \sqsubseteq)$. We will now construct a set S to be shattered by $\mathcal{L}_f$, more precisely, by $\{L_f(w) \mid w \in W\}$.

By Theorem 4.1, L_f is an isomorphism; thus, so is $L_f \circ h$. Hence, for all $m \in M$, $(L_f \circ h)(\{m\})$ and $(L_f \circ h)(M \setminus \{m\})$ are incomparable. Thus, for each $m \in M$, choose $z_m \in (L_f \circ h)(\{m\}) \setminus (L_f \circ h)(M \setminus \{m\})$.

Let $S = \{z_m \mid m \in M\}$. We show that S is shattered by $\mathcal{L}_f$. Let $S' \subseteq S$, and let $M' \subseteq M$ be such that $S' = \{z_m \mid m \in M'\}$. Now, we have $S \cap L_f(h(M')) = S'$ from the properties of L_f and h.

Regarding "$\geq$", let $S \subseteq \Sigma^*$ be shattered by $\mathcal{L}_f$. We show $(\mathrm{Pow}(S), \subseteq)$ is isomorphic to $(\{L_f(\bigsqcup_{x \in D} f(x)) \mid D \subseteq S\}, \subseteq)$. Let h be such that, for all $D \subseteq S$, $h(D) = L_f(\bigsqcup_{x \in D} f(x))$. Obviously, for all D, D' with $D \subseteq D'$ we have $h(D) \subseteq h(D')$. Let D, D' be such that $h(D) \subseteq h(D')$. Let $x \in D$. Clearly, $x \in h(D')$. Let $L \in \mathcal{L}_f$ be such that $S \cap L = D'$. $h(D')$ is the least element of $\mathcal{L}_f$ such that D' is a subset. Thus $h(D') \subseteq L$. We have $x \in S \cap h(D') \subseteq S \cap L = D'$. □

For any upper semi-lattice V, we denote with $\mathrm{width}(V)$ the supremum of the cardinalities of all sets of mutually incomparable elements of V. The following corollary provides an easy-to-check upper bound for the VC-dimension.

Corollary 7.2. *Let (V, f) be an LS. We have*

$$\mathrm{VC}(\mathcal{L}_f) \in O(\log \mathrm{width}(V)).$$

Proof. Theorem 7.1 shows that infinite VC-dimension gives infinite width. Otherwise, suppose the VC-dimension is finite. Using Theorem 7.1, let S be a set of size $\mathrm{VC}(\mathcal{L}_f)$ such that $(\mathrm{Pow}(S), \subseteq)$ can be embedded into V thus, the width of V is at least the width of $(\mathrm{Pow}(S), \subseteq)$, which is $\binom{\mathrm{VC}(V)}{\lfloor \mathrm{VC}(V)/2 \rfloor}$, according to Sperner's Theorem [48]. From well-known formulas for binomial coefficients, we get

$$\mathrm{VC}(\mathcal{L}_f) \in O\left(\log\left(\frac{2^{\mathrm{VC}(V)}}{\sqrt{\mathrm{VC}(V)}}\right)\right) \subseteq O(\log \mathrm{width}(V)). \quad \square$$

Note that the (logarithm of the) width of V can be an arbitrarily bad upper bound on the VC-dimension of an LC. Take, for example, V to be the lattice with a maximum $\top$, minimum $\bot$, and infinitely many incomparable elements in between, one for each element of Σ^*. The resulting LC has infinite width, and VC-dimension 1.

There are infinite LCs with finite width and thus finite VC-dimension. We mention one that relates to the Parikh map [40] and the PAC-learning rays example in [1]. For all $w \in \Sigma^*$, let $|w|_a$ denote the number of *a*s in w (e.g. $|babbba|_a = 2$). Consider the lattice $V = (\mathbb{N}, \leq)$ so that $\bot = 0$ and the function $f_a : \Sigma^* \to \mathbb{N}$ such that $x \mapsto \{n \in \mathbb{N} \mid |x|_a = n\}$. It is easy to see that $\mathcal{L}_{f_a}$ contains infinitely many languages and that the width of this lattice is 1. Each $n \in \mathbb{N}$ corresponds to a language in $\mathcal{L}_{f_a}$: the one that accepts all words that have at most n a's.[17]

8. Conclusions and outlook

Lattice Classes are learnable and are very natural as evidenced by their various characterizations and their attractive properties. It is also striking that so many disparate language classes are Lattice Classes. These include several important classes in the subregular hierarchies, function-distinguishable classes such as the k-reversible languages, and the pattern languages. In some cases, these new insight may be pushed further. For example, lattice-structured classes may bear interesting relations with varieties of languages [14,15] or many other refinements of varieties [41,18].

Furthermore, Lattice Classes are modular, in the sense that the element-wise intersection of two lattice classes is also a lattice class. In domains where there are several different kinds of constraints on the patterns to be learned (e.g. they are both pattern languages and Strictly k-Local languages), Lattice Classes provide a natural way to compose learners.

It is obvious that if f and $\sqcup$ are computable in polytime then lattice learners learn lattice classes in polynomial update time. It is an interesting, open question whether we can say something about the converse as well. In general, this lattice learning approach would probably yield worse learners than algorithms tailored to a given learning task.

Finally, we believe the ideas here can be fruitfully applied to characterizing classes of stochastic languages that can be learned in similar ways.

Acknowledgments

The authors are thankful for the many comments that reviewers of this paper suggested; they helped to significantly improve the paper.

Jeffrey Heinz was supported by the United States National Science Foundation (NSF) under grant no. 1035577.

Timo Kötzing was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant no. NE 1182/5-1.

References

[1] M. Anthony, N. Biggs, Computational Learning Theory, Cambridge University Press, 1992.
[2] D. Angluin, Finding patterns common to a set of strings, Journal of Computer and System Sciences 21 (1980) 46–62.
[3] D. Angluin, Inductive inference of formal languages from positive data, Information and Control 45 (1980) 117–135.
[4] D. Angluin, Inference of reversible languages, Journal of the ACM 29 (1982) 741–765.
[5] D. Angluin, Learning regular sets from queries and counter-examples, Information and Computation 75 (1987) 87–106.
[6] L. Blum, M. Blum, Toward a mathematical theory of inductive inference, Information and Control 28 (1975) 125–155.
[7] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, Learnability and the Vapnik–Chervonenkis dimension, Journal of the ACM 36 (1989) 929–965.
[8] G. Birkhoff, Lattice Theory, American Mathematical Society, 1984.
[9] A. Clark, R. Eyraud, Polynomial identification in the limit of substitutable context-free languages, Journal of Machine Learning Research 8 (2007) 1725–1745.
[10] J. Case, S. Jain, S. Lange, T. Zeugmann, Incremental concept learning for bounded data mining, Information and Computation 152 (1999) 74–110.
[11] J. Case, T. Kötzing, Difficulties in forcing fairness of polynomial time inductive inference, in: Proc. of Algorithmic Learning Theory, 2009, pp. 263–277.
[12] M. de Brecht, M. Kobayashi, H. Tokunaga, A. Yamamoto, Inferability of closed set systems from positive data, in: Proc. of the conference of the JSAI, 2006, pp. 265–275.
[13] C. de la Higuera, Grammatical Inference, Cambridge University Press, 2010.
[14] S. Eilenberg, Automata, Languages and Machines, Vol. A, Academic Press, New York, 1974.
[15] S. Eilenberg, Automata, Languages and Machines, Vol. B, Academic Press, New York, 1976.
[16] H. Fernau, Identification of function distinguishable languages, Theoretical Computer Science 290 (2003) 1679–1711.
[17] M. Fulk, A study of inductive inference machines, Ph.D. Thesis, SUNY at Buffalo, 1985.
[18] Mai Gehrke, Serge Grigorieff, Jean-Éric Pin, Duality and equational theory of regular languages, in: Lect. Notes Comp. Sci, Springer, 2008, pp. 246–257.
[19] C.R. Gallistel, A.P. King, Memory and the Computational Brain, Wiley-Blackwell, 2009.
[20] E. Gold, Language identification in the limit, Information and Control 10 (1967) 447–474.
[21] P. García, J. Ruiz, Learning k-testable and k-piecewise testable languages from positive data, Grammars 7 (2004) 125–140.
[22] P. Garcia, E. Vidal, J. Oncina, Learning locally testable languages in the strict sense, in: Proc. of the Workshop on Algorithmic Learning Theory, 1990, pp. 325–338.
[23] J. Heinz, The inductive learning of phonotactic patterns, Ph.D. Thesis, Linguistics Department, University of California, Los Angeles, 2007.
[24] J. Heinz, Learning long-distance phonotactics, Linguistic Inquiry 41 (2010) 623–661.
[25] J. Heinz, String extension learning, in: Proc. of the Meeting of the Association for Computational Linguistics, 2010, pp. 897–906.
[26] Klaus P. Jantke, Monotonic and nonmonotonic inductive inference of functions and patterns, in: Proc. of the Workshop on Nonmonotonic and Inductive Logic, 1991, pp. 161–177.
[27] S. Jain, S. Lange, S. Zilles, Some natural conditions on incremental learning, Information and Computation 205 (11) (2007) 1671–1684.

[17] Inverting the lattice so $V = (\mathbb{N}, \geq)$ and so $0 = \top$ yields languages that accept at least n *a*s.

[28] S. Jain, D. Osherson, J. Royer, A. Sharma, Systems that Learn: An Introduction to Learning Theory, second ed., MIT Press, 1999.
[29] L. Kari, A. Mateescu, G. Păun, A. Salomaa, Multi-pattern languages, Theoretical Computer Science 141 (1995) 253–268.
[30] S. Kobayashi, Approximate identification, finite elasticity and lattice structure of hypothesis space, Technical Report CSIM 96-04, Dept. of Compt. Sci. and Inform. Math., Univ. of Electro-Communications, 1996.
[31] S. Kobayashi, T. Yokomori, On approximately identifying concept classes in the limit, in: Proc. of Algorithmic Learning Theory, 1995, pp. 298–312.
[32] S. Lange, R. Wiehagen, Polynomial time inference of arbitrary pattern languages, New Generation Computing 8 (1991) 361–370.
[33] S. Lange, T. Zeugmann, Monotonic versus non–monotonic language learning, in: Proc. of the Workshop on Nonmonotonic and Inductive Logic, 1993, pp. 254–269.
[34] S. Lange, T. Zeugmann, S. Zilles, Learning indexed families of recursive languages from positive data: a survey, Theoretical Computer Science 397 (2008) 194–232.
[35] T. Mitchell, Machine Learning, McGraw Hill, 1997.
[36] R. McNaughton, S. Papert, Counter-Free Automata, MIT Press, 1971.
[37] J. Nation, Notes on Lattice Theory, 2009, http://www.math.hawaii.edu/~jb/books.html.
[38] J. Oncina, P. García, E. Vidal, Learning subsequential transducers for pattern recognition tasks, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (1993) 448–458.
[39] D. Osherson, M. Stob, S. Weinstein, Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists, MIT Press, 1986.
[40] R.J. Parikh, On context-free languages, Journal of the ACM 13 (1966) 570–581.
[41] Jean-Eric Pin, A variety theorem without complementation, Russian Mathematics (Izvestija vuzov Matematika) 39 (1995) 74–83.
[42] J. Royer, J. Case, Subrecursive Programming Systems: Complexity and Succinctness, Birkhäuser, Boston, 1994.
[43] J. Rogers, J. Heinz, G. Bailey, M. Edlefsen, M. Visscher, D. Wellcome, S. Wibel, On languages piecewise testable in the strict sense, in: The Mathematics of Language, Springer, 2010.
[44] H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw Hill, New York, 1967, Reprinted by MIT Press, Cambridge, Massachusetts, 1987.
[45] J. Rogers, G. Pullum, Aural pattern recognition experiments and the subregular hierarchy, Journal of Logic, Language and Information 20 (2011) 329–342.
[46] T. Shinohara, Rich classes inferable from positive data: length-bounded elementary formal systems, Information and Computation 108 (1994) 175–186.
[47] I. Simon, Piecewise testable events, in: Automata Theory and Formal Languages, 1975, pp. 214–222.
[48] E. Sperner, Ein Satz über Untermengen einer endlichen Menge, Mathematische Zeitschrift 27 (1928) 544–548.
[49] L. Valiant, A theory of the learnable, Communications of the ACM 27 (1984) 1134–1142.
[50] V.N. Vapnik, A.Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, Theory of Probability and its Applications 16 (1971) 264–280.
[51] K. Wexler, P. Culicover, Formal Principles of Language Acquisition, MIT Press, 1980.
[52] R. Wiehagen, Limes-Erkennung rekursiver Funktionen durch spezielle Strategien, Elektronische Informationverarbeitung und Kybernetik 12 (1976) 93–99.
[53] K. Wright, Identification of unions of languages drawn from an identifiable class, in: Proc. of the Workshop on Computational Learning Theory, 1989, pp. 123–132.
[54] T. Yokomori, Polynomial-time identification of very simple grammars from positive data, Theoretical Computer Science 298 (2003) 179–206.