String Extension Learning Using Lattices

Anna Kasprzik[1], Timo Kötzing[2]

[1] FB IV – Abteilung Informatik, Universität Trier, 54286 Trier, Germany
kasprzik@informatik.uni-trier.de
[2] Department 1: Algorithms and Complexity, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany koetzing@mpi-inf.mpg.de

Abstract. The class of regular languages is not identifiable from positive data in Gold's language learning model. Many attempts have been made to define interesting classes that *are* learnable in this model, preferably with the associated learner having certain advantageous properties. Heinz '09 presents a set of language classes called *String Extension (Learning) Classes*, and shows it to have several desirable properties.
In the present paper, we extend the notion of String Extension Classes by basing it on *lattices* and formally establish further useful properties resulting from this extension. Using lattices enables us to cover a larger range of language classes including the *pattern languages*, as well as to give various ways of *characterizing* String Extension Classes and its learners. We believe this paper to show that String Extension Classes are learnable in a *very natural way*, and thus worthy of further study.

1 Introduction

In this paper, we are mostly concerned with learning as defined by Gold [Gol67] which is sometimes called *learning in the limit from positive data*.

Formally, for a class of (computably enumerable) languages $\mathcal{L}$ and an algorithmic learning function h, we say that h *TxtEx-learns* $\mathcal{L}$ [Gol67, JORS99] iff, for each $L \in \mathcal{L}$, for every function T enumerating (or presenting) all and only the elements of L, as h is fed the succession of values $T(0), T(1), \ldots$, it outputs a corresponding succession of programs $p(0), p(1), \ldots$ from some hypothesis space, and, for some i_0, for all $i \geq i_0$, $p(i)$ is a correct program for L, and $p(i+1) = p(i)$. The function T is called a *text* or *presentation* for L.

There are two main viewpoints in research on language learning: Inductive Inference (II) and Grammatical Inference (GI). The area of Inductive Inference is mainly concerned with the question if a certain target concept, which in our case usually represents a formal language class, can be identified in the limit, i.e., after any finite number of steps. The area of Grammatical Inference is mainly concerned with the concrete algorithms solving that task and with their efficiency, i.e., with the question if the number of steps needed can be bounded by some polynomial with respect to a relevant measure such as the input size or the number of queries asked, if admissible. As a result, research on GI is more involved with the task of inferring a specific description of a formal language (e.g., a grammar or an automaton) than just the language as an abstract item as the inference strategy of any concrete learning algorithm is intrinsically linked to the description it yields as output (for an overview of GI, see [dlH10]). In this paper, we have tried to include results of importance from both perspectives.

Gold [Gol67] already showed that the class of regular languages is *not* TxtEx-learnable. Several papers, for example [Fer03, Hei09], are concerned with finding interesting classes of languages that *are* TxtEx-learnable. Furthermore, frequently it is desirable for a learner to have additional properties, and one wants to find interesting classes learnable by a learner having these properties.

In this paper, we extend and analyze the notion of *String Extension Learning* as given in [Hei09]. We do this by applying Birkhoff-style lattice theory and require all conjectures a learner makes to be drawn from a lattice. Section 2 makes String Extension Learning precise. Importantly, in Theorem 6 we show the resulting learners to have a long list of advantageous properties.

Many simple, but also several more complex languages classes are learnable this way. Some examples are given in [Hei09]; we show in Section 3 how *Pattern Languages* can be learned as a subclass of a String Extension Language Class. Furthermore, Section 3 discusses in what respect *Distinction Languages* [Fer03] are String Extension Languages as well.

Section 4 analyzes String Extension Learners (SELs) and String Extension Classes (SECs) further. We give *two* insightful characterization of SELs in Theorem 14, and *three* characterizations of SECs in Theorem 15. This establishes the SECs as very naturally arising learnable language classes.

Section 5 studies how String Extension Classes and special cases thereof are learnable via *queries* [Ang87]. Complexity issues are discussed and it is shown that String Extension Languages can be learned in a particularly straightforward way from *equivalence queries*.

Familiarity with lattice theory is useful to understand this paper, but not completely necessary. For introductions into lattice theory, the reader is referred to the textbooks [Bir84] (a classic) and [Nat09] (available online).

We omit many proofs due to space constraints. The proof of Theorem 6 is given below and exemplary for several of the omitted proofs. A complete version can be found at `http://www.mpi-inf.mpg.de/~koetzing/StringExtensionLearnersTR.pdf`.

2 Definitions and Basic Properties

Any unexplained complexity-theoretic notions are from [RC94]. All unexplained general computability-theoretic notions are from [Rog67].

$\mathbb{N}$ 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 with Σ^* we denote 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 with ε and the length of a word x with $|x|$.

With $\mathbb{S}\mathrm{eq}$ we denote the set of finite sequences over $\Sigma^* \cup \{\#\}$, where $\#$ is a special symbol called "pause". We denote the empty sequence with $\emptyset$. For a non-empty sequence σ, we let σ^- be the sequence of σ without the last element of σ, and we let $\mathrm{last}(\sigma)$ be the last element of σ. Concatenation on sequences is denoted with $\diamond$. For all $\sigma \in \Sigma^*$, we let $\mathrm{content}(\sigma) = \{x \in \Sigma^* \mid \exists i < \mathrm{len}(\sigma) : \sigma(i) = x\}$.

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets. For sets A, B, we let $A \setminus B = \{a \in A \mid a \notin B\}$, $\overline{A}$ be the complement of A and $\mathrm{Pow}(A)$ be the power set of A.

The quantifier $\forall^\infty x$ means "for all but finitely many x", the quantifier $\exists^\infty x$ means "for infinitely many x". For any set A, $|A|$ denotes the cardinality of A.

We let dom and range denote, respectively, domain and range of a given function. We sometimes denote a function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n \boldsymbol{.} f(x_1, \ldots, x_n)$. For example, with $c \in \mathbb{N}$, $\lambda x \boldsymbol{.} c$ is the constantly c function of one argument.

A function ψ is *partial computable* iff there is a deterministic, multi-tape Turing machine computing ψ. $\mathcal{P}$ and $\mathcal{R}$ denote, respectively, the set of all partial computable and the set of all total (partial) computable functions $\mathbb{N} \to \mathbb{N}$. We say that ψ is *polytime* iff ψ is computable in polynomial time. If a function f is defined for $x \in \mathrm{dom}(f)$ we write $f(x)\downarrow$, and we say that f on x *converges*.

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

We say that a function f *converges to* p iff $\forall^\infty x : f(x)\downarrow = p$.

Whenever we consider (partial) computable functions on objects like finite sequences or finite sets, we assume those objects to be efficiently coded as natural numbers. We also assume words to be so coded. The size of any such finite object is the size of its code number.

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

String Extension Learning

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

Definition 1. A pair $(V, \sqsubseteq)$ is a *partially ordered set* iff

- $\forall a, b : a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b$;
- $\forall a, b : a \sqsubseteq a$;
- $\forall a, b, c : a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$.

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 $\forall a \in S : a \sqsubseteq v$;
- an *lower bound of* S iff $\forall a \in S : v \sqsubseteq a$;
- a *maximum of* S iff v is upper bound of S and $v \in S$;
- a *minimum of* S iff v is 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 a supremum and an infimum.

In an upper semi-lattice, the supremum of two elements $a, b \in V$ is denoted by $a \sqcup b$ and we use $\bigsqcup$ to denote suprema of sets D (note that, in an upper semi-lattice, each non-empty finite set has a supremum, which equals the iterated supremum of its elements, as the binary supremum is an associative operation); if V has a minimum element, then, by convention, $\bigsqcup \emptyset = \bot_V$. In a lower semi-lattice, the infimum of two elements $a, b \in V$ is denoted by $a \sqcap b$.

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

Let V be a partially ordered set with minimum element. An element $a \in V$ is called an *atom* iff $a \neq \bot_V$ and $\{b \mid \bot_V \sqsubseteq b \sqsubseteq a\} = \{\bot_V, a\}$. If, for all $b \in V$ there is an atom $a \in V$ such that $a \sqsubseteq b$, then we call V *atomic*.

A lattice is called *boolean* iff V has a minimal element $\bot$ and maximal element $\top$ and there is a function $\bar{\cdot}$ such that, for all $a \in V$, $a \sqcap \overline{a} = \bot$ and $a \sqcup \overline{a} = \top$.

For example, for each $k \in \mathbb{N}$, the set of all finite sets that contain only words of length k is, with inclusion as the order, an atomic lattice, which is boolean iff the alphabet is finite. We call this lattice V_{fac-k}.

Definition 2. For an upper semi-lattice V and a function $f : \Sigma^* \to V$ such that f and $\sqcap$ are (total) computable, (V, f) is called a *String Extension Space (SES)* iff, for each $v \in V$, there is a finite $D \subseteq \text{range}(f)$ with $\bigsqcup_{x \in D} x = v$.[3]

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

As an example, for each k, we let $fac_k : \Sigma^* \to V_{fac-k}, x \mapsto \{v \in \Sigma^k \mid \exists u, w \in \Sigma^* : x = uvw\}$. Then (V_{fac-k}, fac_k) is an SES.[4]

Definition 3. Let (V, f) be an SES.

- A *grammar* is any $v \in V$.[5]

[3] This definition might seem a little strange at first, and in general one could define SESes without those restrictions. However, elements that are not the finite union 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" SESes in the first place.

[4] Any substring of length k of a word x is called a *k-factor* of x.

[5] 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.

– 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 $\mathcal{L}_f = \{L_f(v) \mid v \in V\}$.

We define ϕ_f such that $\forall v, x : \phi_f(\sigma) = \bigsqcup_{x \in \mathrm{content}(\sigma)} f(x)$.

Any class of languages $\mathcal{L}$ such that there is an SES (V, f) with $\mathcal{L} = \mathcal{L}_f$ is called a *String Extension Class (SEC)*, ϕ_f a *String Extension Learner (SEL)*.[6] We will omit the subscript of f if it is clear from context.

For example, with respect to (V_{fac-2}, fac_2), $\{aa, ab\}$ is a grammar for the set of all words for which any contiguous subword of length 2 is either aa or ab. Example such words include aaa, ab, $aaaab$, c, ...

Next we define what we mean by "learning".

Definition 4. Let $L \subseteq \Sigma^*$ and $T : \mathbb{N} \to \Sigma^*$. T is called a *text for* L iff $L \subseteq \mathrm{content}(T) \subseteq L \cup \{\#\}$. Let $h : \mathbb{S}\mathrm{eq} \to \mathbb{N}$ be a (total computable) learner. We assume the outputs of h to be mapped by a function $L(\cdot)$ to a language. Whenever no concrete function $L(\cdot)$ is given, we assume the mapping $\lambda p \centerdot W_p$.

The learner $h \in \mathcal{P}$ is said to **TxtEx**-*identify* a languages L with respect to $L(\cdot)$ 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 $\mathrm{Conv}(h, T) = k$.

We denote the set of all languages **TxtEx**-*identified* by a learner h with **TxtEx**(h). We say that a class of languages $\mathcal{L}$ is **TxtEx**-*identified* (possibly with certain properties) iff there is a learner $h \in \mathcal{P}$ (observing those properties) **TxtEx**-learning every set in $\mathcal{L}$. Further, we say that h learns a language using a uniformly decidable hypothesis space iff $\lambda x, p \centerdot x \in L(p)$ is (total) computable.

The following learner properties have been studied in the literature.

Definition 5. Let a learner $h : \mathbb{S}\mathrm{eq} \to \mathbb{N}$ be given. We call h

– iterative [Ful85, Wie76], 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 a polytime such function h^{it};
– set-driven [WC80, JORS99], iff there is a function $h^{set} : \mathrm{Pow}(\Sigma^*) \to \mathbb{N}$ such that $\forall \sigma \in \mathbb{S}\mathrm{eq} : h^{set}(\mathrm{content}(\sigma)) = h(\sigma)$;
– globally consistent [Bār74, BB75, Wie76], iff $\forall \sigma \in \mathbb{S}\mathrm{eq} : \mathrm{content}(\sigma) \subseteq L(h(\sigma))$;
– locally conservative [Ang80], iff $\forall \sigma \in \mathbb{S}\mathrm{eq}, x \in \Sigma^* : h(\sigma) \neq h(\sigma \diamond x) \Rightarrow x \notin L(h(\sigma))$;

[6] In general, 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 10.

– strongly monotone [Jan91], iff $\forall \sigma \in \mathbb{S}\mathrm{eq}, x \in \Sigma^* : L(h(\sigma)) \subseteq L(h(\sigma \diamond x))$;
– prudent [Wei82, OSW86], iff $\forall \sigma \in \mathbb{S}\mathrm{eq} : L(h(\sigma)) \in \mathbf{TxtEx}(h)$;
– optimal [Gol67], iff, for all learners h' with $\mathbf{TxtEx}(h) \subseteq \mathbf{TxtEx}(h')$,

$$\begin{aligned} &\exists L \in \mathbf{TxtEx}(h), T \in \mathbf{Txt}(L) : \mathrm{Conv}(h', T) < \mathrm{Conv}(h, T) \\ &\qquad\qquad\Rightarrow \\ &\exists L \in \mathbf{TxtEx}(h), T \in \mathbf{Txt}(L) : \mathrm{Conv}(h, T) < \mathrm{Conv}(h', T). \end{aligned} \tag{1}$$

We briefly show that SELs have a number of desirable properties.

Theorem 6. Let (V, f) be an SES. Then ϕ_f **TxtEx**-learns $\mathcal{L}_f$

(i) iteratively;
(ii) if (V, f) is a polytime SES, 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 SES, let $D \subseteq \Sigma^*$ such that $v = \bigsqcup_{x \in D} f(x)$. Let k be such that $D \subseteq \mathrm{content}(T[k])$. 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

$$\forall v, x : \phi_f^{it}(v, x) = \begin{cases} v, & \text{if } x = \#; \\ v \sqcup f(x), & \text{otherwise.} \end{cases} \tag{2}$$

(ii) Clearly, ϕ_f^{it} from (i) is polytime, if (V, f) is a polytime SES.
(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 $x \in \mathrm{content}(\sigma)$. Then $f(x) \sqsubseteq \bigsqcup_{y \in \mathrm{content}(\sigma)} f(y) = \phi_f(\sigma) = v$. Thus, $x \in L(v)$.
(v) Let $\sigma \in \mathbb{S}\mathrm{eq}$ and $x \in \Sigma^*$ with $\phi_f(\sigma) \neq \phi_f(\sigma \diamond x)$. Thus, $\phi_f(\sigma) \neq \phi_f(\sigma) \cup f(x)$, in particular, $f(x) \not\sqsubseteq \phi_f(\sigma)$. Therefore, $x \notin L(\phi_f(\sigma))$.
(vi) Let $\sigma \in \mathbb{S}\mathrm{eq}$ 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 $\sigma \in \mathbb{S}\mathrm{eq}$ 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 [OSW86, Proposition 8.2.2A].

□

For each SES (V, f) we will use ϕ_f^{it} and ϕ_f^{set} as shown existent just above.

3 Example SECs

We already came across the example of k-factor languages and its SES (V_{fac-k}, fac_k). Many more examples like this can be found in [Hei09]. In this section we define a more complex example.

Definition 7. 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 $\pi \in \mathbb{P}\mathrm{at}$, let $L(\pi) =$

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

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 [CJLZ99, Corollary 12]. The usual iterative algorithm is first published in [LW91].

Theorem 8. For any finite set $D \subseteq \Sigma^*$, we let $\mathrm{pat}(D) = \bigcap_{w \in D} \mathrm{pat}(w)$.[7] Let V_{pat} be the lattice $\{\mathrm{pat}(D) \mid D \subseteq \Sigma^* \text{ finite}\}$ with order relation $\supseteq$.[8] Then (V_{pat}, pat) is an SES.

Now ϕ_{pat} learns the pattern languages maximally consistently and iteratively (as well as with all other properties as given in Theorem 6). Note that some of the grammars of (V_{pat}, pat) are not for pattern languages, for example $\mathrm{pat}(\{a^3, b^4\}) = \{x_1, x_1x_2, x_1x_2x_3, x_1x_1x_2, x_1x_2x_1, x_1x_2x_2\}$.

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

Fernau [Fer03] introduced the notion of *distinguishable languages* (DLs). The following shows that the concept of DLs is subsumed by the concept of SECs, while the concept of SECs is *not* subsumed by the concept of DLs.

Theorem 9.

$$\mathrm{DL} \subset \mathrm{SEC}.$$

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

Proof. "$\neq$": Obviously, the class of all finite languages is an SEL but not a DL.

"$\subseteq$": Let $\mathcal{L}$ be a DL. Let h be the learner for $\mathcal{L}$ given in [Fer03, § 6]. By [Fer03, Theorem 35], h fulfills the condition of Theorem 14(iii). Hence, h is a String Extension Learner by Theorem 14 and $\mathcal{L}$ is an SEC. $\square$

For the reader familiar with [Fer03] we specify a concrete SES (V, f) such that ϕ_f learns the class of f'-DLs for any distinguishing function $f' : \Sigma^* \to X$.

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

[8] Note that the order is inverted with respect to the usual powerset lattice.

Define V as the set of all stripped[9] 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 adding all shortest strings leading to an accepting state in B_0, and then for every hitherto unrepresented transition $\delta(q_1, a) = q_2$ ($a \in \Sigma$) of B_0 adding the string resulting from concatenating a string leading to q_1, a, and a string leading from q_2 to an accepting state. Use the learner h from [Fer03] on I_+. By Lemma 34 and Theorem 35 in [Fer03] 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 [Fer03], Lemma 15). We show that (V, f) is an SES.

Obviously, f is computable. For each $v \in V$ there is a finite set $D \in \text{range}(f)$ 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.[10]

4 Properties of SECs

In this section we give a number of interesting theorems pertaining to SECs and their learnability. Most importantly, we characterize SELs (Theorem 14) and SECs (Theorem 15).

Theorem 10. Let (V, f) be an SES. 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

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

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

[10] 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 incremental version of the learner from [Fer03].

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

$$\{x \in \Sigma^* \mid f(x) \sqsubseteq a\} = L_f(a) \subseteq L_f(b) = \{x \in \Sigma^* \mid f(x) \sqsubseteq b\}. \tag{4}$$

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, $a \sqsubseteq b$. □

Corollary 11. Let (V, f) and (W, g) be two SESes with $\mathcal{L}_f \subseteq \mathcal{L}_g$. Then there is an order embedding $h : V \to W$.

Proof. Define h such that

$$\forall v \in V : L_f(v) = L_g(h(v)). \tag{5}$$

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

$$a \sqsubseteq_V b \Leftrightarrow L_f(a) \subseteq L_f(b) \Leftrightarrow L_g(h(a)) \subseteq L_g(h(b)) \Leftrightarrow h(a) \sqsubseteq h(b). \tag{6}$$

□

Lemma 12. Let (V, f) be an SES. We have the following.

(i) For all $a, b \in V$, $L(a) \cup L(b) \subseteq L(a \sqcup b)$.
(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 V is a lattice, then we have, for all $a, b \in V$, $L(a) \cap L(b) = L(a \sqcap b)$.

Proof.

(i) Let $a, b \in V$, let $x \in L(a) \cup L(b)$. Then $f(x) \sqsubseteq a$ or $f(x) \sqsubseteq b$; thus, $f(x) \sqsubseteq a \sqcup b$. Therefore, $x \in L(a \sqcup b)$.
(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 10, $(\mathcal{L}, \subseteq)$ 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 $a, b \in V$. We have, for all $x \in \Sigma^*$,

$$x \in L(a \sqcap b) \Leftrightarrow f(x) \sqsubseteq a \sqcap b \Leftrightarrow f(x) \sqsubseteq a \text{ and } f(x) \sqsubseteq b \Leftrightarrow x \in L(a) \cap L(b).$$

□

Theorem 13. Let (V, f) be an SES. We have the following.

(i) $\lambda v, x \centerdot x \in L(v)$ is computable (i.e., $(L(v))_{v \in V}$ is uniformly decidable).
(ii) If (V, f) is polytime, then $\lambda v, x \centerdot x \in L(v)$ is computable in polynomial time (i.e., $(L(v))_{v \in V}$ is uniformly decidable in polynomial time).
(iii) $\mathcal{L}$ is closed under intersection iff V is a lattice.

Proof.

(i) We have $\lambda v, x \centerdot [x \in L(v)] = \lambda v, x \centerdot [\phi_f^{it}(v, x) = v]$ by consistency and conservativeness of ϕ_f. Clearly, $\lambda v, x \centerdot [\phi_f^{it}(v, x) = v]$ is computable.
(ii) Using Theorem 6(ii), analogous to the just above proof of (i).
(iii) "$\Rightarrow$": Follows from the isomorphie given in Theorem 10.
"$\Leftarrow$": Follows directly from Lemma 12(iv).

□

Now we get to our main theorem of this section, which shows that all learners having a certain subset of the properties listed above in Theorem 6 can necessarily be expressed as SELs.

Theorem 14. Let $h \in \mathcal{R}$. The following are equivalent.

(i) There is an SES (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 $L(\cdot)$ such that $L(v)$ is $\mathtt{ce}$ uniformly in v and, for all $\sigma \in \mathbb{S}\mathrm{eq}$, $L(h(\sigma))$ is the $\subseteq$-minimum element of $\mathbf{TxtEx}(h)$ containing all of content(σ).

Proof. We have that (i) implies (iii) by basic properties of the SEL.

Regarding (iii) implies (ii): set-drivenness, global consistency and local conservativeness are straightforward. Then h is prudent [CK09, 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 $\mathbf{TxtEx}(h)$ containing all of D (from prudence we have $L(h^{set}(D)) \in \mathbf{TxtEx}(h)$). 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, $h(\sigma) = h^{set}(\text{content}(\sigma))$. Note that, for all D, D' such that $h^{set}(D) = h^{set}(D')$, we have

$$h^{set}(D) = h^{set}(D \cup D') \tag{7}$$

by consistency and conservativeness.

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

$$\forall D_0, D_1 : h^{set}(D_0) \sqcup h^{set}(D_1) = h^{set}(D_0 \cup D_1). \tag{8}$$

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

$$\begin{aligned} & D_0 \cup D_0' \cup D_1 \cup D_1' && (9)\\ \underset{\text{cons.}}{\subseteq}\; & L(h^{set}(D_0)) \cup L(h^{set}(D_0')) \cup L(h^{set}(D_1)) \cup L(h^{set}(D_1')) && (10)\\ =\; & L(h^{set}(D_0)) \cup L(h^{set}(D_1)) && (11)\\ \underset{\text{str. mon.}}{\subseteq}\; & L(h^{set}(D_0 \cup D_1)). && (12)\end{aligned}$$

Similarly, we get

$$D_0 \cup D_0' \cup D_1 \cup D_1' \subseteq L(h^{set}(D_0' \cup D_1')). \qquad (13)$$

From conservativeness we get

$$h^{set}(D_0 \cup D_1) = h^{set}(D_0 \cup D_0' \cup D_1 \cup D_1') = h^{set}(D_0' \cup D_1'). \qquad (14)$$

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 [Nat09, 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 SES as desired. □

Theorem 15. Let $\mathcal{L}$ be a set of languages. The following are equivalent.

(i) $\mathcal{L}$ is an SEC.

(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 $L(v)$ is $\mathtt{ce}$ uniformly in 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 all of 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 14.

Further, (i) implies (iv) by Theorems 6 and 13.

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. We will define a learner $h' \in \mathcal{P}$ using hypotheses in the W-system. We will use functions $p, q \in \mathcal{R}$ as follows to define hypotheses in the W-system.

$$\begin{aligned} \forall e, \sigma : W_{p(e,\sigma)} &= L(e) \cup \mathrm{content}(\sigma); && (15)\\ \forall e : W_{q(e)} &= L(e). && (16)\end{aligned}$$

Employing these functions, we define h' as follows.

$$\forall \sigma \in \mathbb{S}\mathrm{eq} : h'(\sigma) = \begin{cases} p(h(\sigma), \sigma), & \text{if } \mathrm{content}(\sigma) \not\subseteq L(h(\sigma)); \\ q(h'(\sigma^-)), & \text{else if } \sigma \neq \emptyset \wedge \mathrm{content}(\sigma) \subseteq L(h'(\sigma^-)); \\ q(h(\sigma)), & \text{otherwise.} \end{cases} \quad (17)$$

It is easy to see that h' is set-driven, globally consistent, locally conservative and strongly monotonic. □

Proposition 16 just below gives a sufficient condition for a language to be an SEC.

Proposition 16. 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 SEC.

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 maximally 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 identifies L_0, we have, for all $k \geq \mathrm{len}(\sigma)$, $h(T[k]) = h(\sigma)$. Thus, $L(h(\sigma)) = L_0$. This shows the lemma, as

$$L(h(\sigma)) = L_0 = L(h(\sigma)) \cap L(h(\sigma \diamond x)) \subseteq L(h(\sigma \diamond x)). \quad (18)$$

We now get that $\mathcal{L}$ is an SEC by Theorem 15. □

5 Query Learning of SECs

This section is concerned with learning SECs from queries. We address the issue from a more GI-oriented view, inasmuch as for example some concrete algorithms are given and complexity questions are considered.

Definition 17. Let (V, f) be an SES and $v \in V$ the target to identify.[11] 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 MQ $= 1$ if $w \in L$ and MQ $= 0$ otherwise.[12] An *equivalence query (EQ)* for $v_0 \in V$ is a query '$L(v_0) = L(v)$?' receiving an answer from $\Sigma^* \cup \{\texttt{yes}\}$ ($\Sigma^* \cap \{\texttt{yes}\} = \emptyset$) such that $\mathrm{EQ}(v_0) = \texttt{yes}$ for $L(v_0) = L(v)$ and $\mathrm{EQ}(v_0) = c$ with $[f(c) \sqsubseteq v \wedge \neg f(c) \sqsubseteq v_0] \vee [f(c) \sqsubseteq v_0 \wedge \neg f(c) \sqsubseteq v]$ otherwise.

[11] To be precise, the concept to infer is a language. However, as no two elements of V define the same language (see Footnote 6) our potential targets are elements of V.

[12] Algorithmically, using MQs only makes sense if the membership problem is decidable. As for SECs we have $f : \Sigma^* \to V$ an MQ for $w \in \Sigma^*$ amounts to checking if $f(w) \sqsubseteq v$.

Let (V, f) be an SES. As $\mathcal{L}_f$ is identifiable in the limit from text (see [Hei09]) $\mathcal{L}_f$ is also identifiable in the limit from MQs: Consider a learner just querying all strings of Σ^* in length-lexical order – at some point the set of all strings w with $\mathrm{MQ}(w) = 1$ queried sofar necessarily includes a text for the target.

If we are interested in complexity, unfortunately in general we cannot bound the number of MQs needed in any interesting way. For each $v \in V$, define $\mathbb{T}_v := \{T \subseteq \mathrm{range}(f) | \bigsqcup_{t \in T} t = v\}$ and let T_0 be an element of $\mathbb{T}_v$ with minimal cardinality. Obviously, $|T_0|$ is a lower bound on the number of MQs needed to converge on the target. However, note that there exist SECs with properties that allow more specific statements:

Theorem 18. Let (V, f) be an SES. If $\mathcal{L}_f$ is the class of k-factor languages or the class of k-piecewise testable languages (see [Hei09]) identification is possible with a query complexity of $O(|\Sigma|^k)$ MQs.

Proof. We give a simple learning algorithm in pseudo-code that can be used to identify any SEC $\mathcal{L}_f$ such that there is a finite set Q with $\forall v \in V : \exists S \subseteq Q : f(S) \in \mathbb{T}_v$. Observe that we can set $Q := \Sigma^k$ for the k-factor languages and $Q := \Sigma^{\leq k}$ for the k-piecewise testable languages.

```
[Initialize $Q_0$ with the respective $Q$ as given just above];
$v_0 := \perp_V$;
for all $s \in Q_0$ do:
    if MQ($s$) = 1 $v_0 := v_0 \sqcup f(s)$
return $v_0$.
```

It is easy to see that this algorithm yields the target after $|Q_0|$ loop executions which corresponds to having asked $|Q_0|$ MQs, where $|Q_0| = |\Sigma^k| = |\Sigma|^k$ for the k-factor languages and $|Q_0| = |\Sigma^{\leq k}| = (|\Sigma|^{k+1} - 1)/(|\Sigma| - 1)$ such that $O(|Q_0|) = O(|\Sigma|^k)$ for the k-piecewise testable languages. □

Remark: If the SES is polytime it follows from Theorem 18 that in these special cases identification is possible in polytime as well.

However, as stated above, polytime identification cannot be ensured in the general case. The situation changes if we allow EQs instead of MQs:

Theorem 19. Let (V, f) be an SES. Then $\mathcal{L}_f$ is identifiable in the limit from EQs. In particular, for all $v \in V$, if the length of each ascending path from $\perp_V$ to v is at most n then v can be identified using $O(n)$ EQs.

Proof. Let $v \in V$ be the target. We give a concrete learning algorithm:

```
$v_0 := \perp_V$;
repeat until EQ($v_0$) = yes:
    $c$ := EQ($v_0$);
    $v_0 := v_0 \sqcup f(c)$
return $v_0$.
```

Lemma 20. The algorithm identifies any target $v \in V$ using $O(n)$ EQs.

As $v_0 \sqsubseteq v$ before each loop execution counterexample $c = \mathrm{EQ}(v_0)$ must be chosen such that $f(c) \sqsubseteq v$, which entails $v_0 \sqcup f(c) \sqsubseteq v$. The fact that c is a counterexample implies $\neg(f(c) \sqsubseteq v_0)$ and $v_0 \sqcup f(c) \neq v_0$. Consequently, the successive values of v_0 in the execution of the algorithm form a path from $\perp_V$ to v, where the length of this path equals the number of loop executions. Hence, the target v is identified in finitely many steps using $O(n+1) = O(n)$ EQs (which corresponds to receiving at most n counterexamples). $\square$

If n can be bounded by some polynomial relating the length of the longest ascending path from $\perp_V$ to v to the size of the grammar, and if (V, f) is polytime, then $\mathcal{L}_f$ is identifiable in polytime from EQs as well. Remark: EQs as such cost as much as it costs the teacher to compare two elements of the lattice.

EQs have two advantages: First, the learner actually *knows* when he has identified the target, namely when the teacher has no more counterexamples to give and answers the next EQ in the positive. And second, unlike in the general cases of learning from text or MQs where the learner has to handle strings $s \in \Sigma^* \setminus L_f(v)$ that do not change the current hypothesis at all, EQs can be used in such a way that every EQ results in the retrieval of at least one more hitherto unrevealed element of the target (which can be seen from the fact that we "make progress" in every loop execution of the algorithm given above).

6 Conclusion and Outlook

We have given a general definition of String Extension Classes and have shown that several natural examples are SECs. We have argued further for the naturality of these language classes by giving various characterizations and properties.

It seems to us that the lattice theoretic framework can be highly beneficial to the analysis of *classes* of learning algorithms. For example, one can analyze the probabilistic learnability of String Extension Classes – we have some promising preliminary results pertaining to atomic lattices.

Furthermore, one could try to find other natural language classes characterized by a different kind of learnability, for example by dropping the requirement of strong monotonicity and possibly picking up some other requirement.

References

[Ang80] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[Ang87] D. Angluin. Learning regular sets from queries and counter-examples. *Information and Computation*, 75:87–106, 1987.

[Bār74] J. Bārzdiņš. Inductive inference of automata, functions and programs. In *Proceedings of the 20th International Congress of Mathematicians, Vancouver, Canada*, pages 455–560, 1974. English translation in, *American Mathematical Society Translations*: Series 2 109 (1977), pp. 107-112.

[BB75] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[Bir84] G. Birkhoff. *Lattice Theory.* American Mathematical Society, Providence, RI, 1984.

[CJLZ99] J. Case, S. Jain, S. Lange, and T. Zeugmann. Incremental concept learning for bounded data mining. *Information and Computation*, 152:74–110, 1999.

[CK09] John Case and Timo Kötzing. Difficulties in forcing fairness of polynomial time inductive inference. In *ALT*, volume 5809 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009.

[dlH10] C. de la Higuera. *Grammatical Inference.* Cambridge University Press, 2010. In press.

[Fer03] H. Fernau. Identification of function distinguishable languages. *Theoretical Computer Science*, 290(3), 2003.

[Ful85] M. Fulk. *A Study of Inductive Inference Machines.* PhD thesis, SUNY at Buffalo, 1985.

[Gol67] E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Hei09] J. Heinz. String extension learning, 2009. http://phonology.cogsci.udel.edu/~heinz/papers/heinz-sel.pdf.

[Jan91] K. P. Jantke. Monotonic and non-monotonic inductive inference. *New Generation Computing*, 8(4), 1991.

[JORS99] S. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that Learn: An Introduction to Learning Theory.* MIT Press, Cambridge, Mass., second edition, 1999.

[LW91] S. Lange and R. Wiehagen. Polynomial time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

[Nat09] J. Nation. Notes on lattice theory, 2009. http://www.math.hawaii.edu/~jb/books.html.

[OSW86] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, Cambridge, Mass., 1986.

[RC94] J. Royer and J. Case. *Subrecursive Programming Systems: Complexity and Succinctness.* Research monograph in *Progress in Theoretical Computer Science*. Birkhäuser Boston, 1994.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw Hill, New York, 1967. Reprinted by MIT Press, Cambridge, Massachusetts, 1987.

[WC80] K. Wexler and P. Culicover. *Formal Principles of Language Acquisition.* MIT Press, Cambridge, Mass, 1980.

[Wei82] S. Weinstein, 1982. Private communication at the *Workshop on Learnability Theory and Linguistics*, University of Western Ontario.

[Wie76] R. Wiehagen. Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. *Elektronische Informationverarbeitung und Kybernetik*, 12:93–99, 1976.