Control Structures in Hypothesis Spaces: The Influence on Learning

John Case [a], Sanjay Jain [b], Mandayam Suraj [a]

[a] *Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA. Email: {case,suraj}@cis.udel.edu*

[b] *School of Computing, National University of Singapore, Singapore 117543, Republic of Singapore. Email: sanjay@comp.nus.edu.sg*

Abstract

In any learnability setting, hypotheses are conjectured *from some hypothesis space*. Studied herein are the influence on learnability of the *presence or absence of certain control structures* in the hypothesis space. First presented are control structure *characterizations* of some rather specific but illustrative learnability results. The presence of these control structures is thereby shown essential to maintain full learning power. Then presented are the main theorems. Each of these non-trivially characterizes the invariance of a learning class over hypothesis space V *and* the presence of a particular projection control structure, called **proj**, in V as: V has suitable instances of *all* denotational control structures. In a sense, then, **proj** epitomizes the control structures whose *presence* needn't help and whose *absence* needn't hinder learning power.

Keywords: Computational Learning Theory, Inductive Inference, R.E. Languages, Numberings, Control Structures.

1 Introduction

In any learnability setting, hypotheses are conjectured *from some hypothesis space*, for example, in [JORS99] from general purpose programming systems, in [ZL95,Wie78] from subrecursive systems, and in [Qui93] from very simple classes of classificatory decision trees. For example, with the latter one can, nonetheless, train an autopilot from flight simulator data on real pilots [CSM92]. Much is known theoretically about the restrictions on learning power resulting from restricted hypothesis spaces [ZL95].

In the present paper we begin to study the influence on learnability of the *presence or absence of certain control structures* in the hypothesis space. We consider herein general purpose systems V for the entire class of r.e. languages, which systems may or may not have available particular control structures. [BS94] considered, in effect, whether a particular learnability result **P** characterized the general purpose hypothesis spaces having available all possible control structures; they discovered their particular **P** failed very badly to do so. We began our study with the idea in mind of seeing if certain control structures (in general purpose systems) were necessary and sufficient to maintain the invariance (compared with a system with all possible control structures available) of standard learning classes. We haven't quite achieved that, and our paper is an initial progress report on the endeavor. ([Wie96] quite interestingly characterizes learning criteria invariances, but as in [Wie78,FKW84], not in terms of control structures.)

In Section 2.1 we present the basics of the sorts of general purpose recognizing systems we consider. We treat (see Section 2.2) mostly the standard learning criteria of learning in the limit and learning in one-shot, recognizers (or grammars [HU79,Wei87]) for r.e. languages — from text (or positive information). In Section 2.3 we provide sufficient background material from [Ric80,Ric81,Roy87] about control structures in general purpose programming systems.

In Section 3 we first present control structure *characterizations* of some rather specific but illustrative learnability results. The presence of these control structures is thereby shown essential to maintain full learning power. In the remainder of this section we consider, for the control structures involved, whether or not they are available in any hypothesis space.

In Section 4 we present our two main characterization theorems, Theorems 39 and 40. Each, essentially, non-trivially characterizes the invariance of a learning class over hypothesis space V *and* the presence of a particular projection control structure, called **proj**, in V as: V has suitable instances of *all* denotational control structures. In a sense, then, **proj** epitomizes the control structures whose *presence* needn't help and whose *absence* needn't hinder learning power. Some parts of these theorems are the most difficult in the paper, namely, the independence of the presence of **proj** from the learning class invariances.

Lastly in Section 5 we present some conclusions, problems, and future directions.

2 Notations and Definitions

We let N denote the set of natural numbers, i.e., $\{0, 1, 2, 3, \ldots\}$. We let lower case math font letters (except d, f, g, h, t), with or without decorations (decorations are the subscripts, superscripts, and the like), range over N. $\emptyset$ denotes the empty set. 2^N denotes the set of all subsets of N. $\in, \notin, \subseteq, \subset$ respectively denote 'is a member of', 'is not a member of', 'is a subset of' and 'is a proper subset of'. For sets A and B, $A \oplus B = (\{2 \cdot x \mid x \in A\} \cup \{2 \cdot x + 1 \mid x \in B\})$ [Rog67]. When iterating the $\oplus$ operator, we will assume left-associativity (to avoid excessive parenthesization). For S, a subset of N, $\mathrm{card}(S)$ denotes the cardinality of S. $\max(S)$ and $\min(S)$ denote, respectively, the maximum and minimum of the set S, where $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$. D_x denotes the finite set with canonical index x [Rog67]. $\langle \cdot, \cdot \rangle$ denotes a fixed *pairing function* [Rog67], a computable, surjective and injective mapping from $N \times N$ into N. $\langle \cdot, \cdot \rangle$ is useful, for example, for speaking of two inputs to a one-input program. d, f, g, h and t with or without decorations range over *total* (not necessarily computable) functions with arguments and values from N.

Let φ_p be the partial computable function: $N \to N$ computed (according to some standard I/O conventions) by Turing machine number p in some standard numbering of Turing machines [Rog58,Rog67,Ric80,Ric81,Roy87]. Let W_p denote the domain of φ_p. Then W_p is the set *recognized* [HU79,Wei87] by Turing machine number p, i.e., the set of natural number inputs on which Turing machine p halts. Let Φ denote a step-counting Blum complexity measure for φ_p [Blu67,DSW94]. We let

$$\varphi_{p,s}(x) = \begin{cases} \varphi_p(x) & \text{if } x \leq s \text{ and } \Phi_p(x) \leq s; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We then let $W_{p,s}$ be the domain of $\varphi_{p,s}$.

The set of all recursively enumerable languages is denoted by $\mathcal{E}$. L and S, with or without decorations, range over $\mathcal{E}$. $\overline{L}$ denotes the complement of L. $\mathcal{L}$, with or without decorations, ranges over subsets of $\mathcal{E}$. For a set L, we use χ_L to denote the characteristic function of L, the function which is 1 on L and 0 off L. $\overline{L}$ denotes complement of L, i.e., $N - L$.

The quantifiers '$\stackrel{\infty}{\forall}$', and '$\stackrel{\infty}{\exists}$' essentially from [Blu67], mean 'for all but finitely many' and 'there exist infinitely many', respectively.

We next define a *limiting-computable function*. For this, we first define

$$\lim_{t \to \infty} h(x,t) = \begin{cases} y & \text{if } (\stackrel{\infty}{\forall} t)[h(x,t) = y]; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We write $h(x,\infty)$ for $\lim_{t\to\infty} h(x,t)$. Function $g : N \to N$ is *limiting-computable* iff $(\exists$ computable $h : (N \times N) \to N)(\forall x)[g(x) = h(x,\infty)]$.

Intuitively, $h(x,t)$ is the output at discrete time t of a mind changing algorithm for g (acting on input x); hence, for g limiting computable as just above, for all x, for all but finitely many times t, the output of the mind changing algorithm on input x is $g(x)$.

In this paper we freely use Church's lambda notation [Chu41,Rog67,All78] to define functions: $N \to N$. For example, $\lambda x \centerdot x+1$ denotes the function that maps each $x \in N$ to $x+1$.

2.1 Computable Recognizing Systems

As we noted in Section 1, in any learnability setting, hypotheses are conjectured *from some hypothesis space*. Furthermore, we noted that in the present paper we focus our attention on hypothesis spaces for recognizing the entire class of r.e. sets. The collection of Turing machines (or their code numbers) defining the sets W_p, $p = 0,1,2,\ldots$ (from Section 2 above) forms such an hypothesis space. We write W as the name of this particular hypothesis space. Of course Turing machines have a universal interpreter which is also a Turing machine. We are also interested in the present paper in focusing our attention on hypothesis spaces containing a universal interpreter for the hypothesis space. Formally this can be handled as follows, where for mappings V in this definition, we write V_p for the value of the mapping V at p.

Definition 1 V is a *computable recognizing system* (abbreviated: *c.r.s.*) iff $V : N \stackrel{\text{onto}}{\to} \mathcal{E}$ such that for some computable t, for every p, $V_p = W_{t(p)}$

Intuitively, for a c.r.s. V, each r.e. set is some V_p, and we have some uniform computable way to map any V-recognizer p into a corresponding Turing machine recognizer $t(p)$ which recognizes the set V_p.

Definition 2 Suppose V is a c.r.s. For L r.e., $\mathrm{MinGram}_V(L)$ denotes $\min(\{p \mid V_p = L\})$.

We define next some interesting senses in which one can translate from one c.r.s. into another. Part (b) of this definition is based on a definition in [Rog58]. [ZL95] notes the relevance to learning theory of the sense in part (c).

Definition 3 Suppose V^1 and V^2 are c.r.s.'s

(a) We say that t *translates* V^1 *into* V^2 (written: $t : V^1 \leq V^2$) iff $(\forall p)[V^2_{t(p)} = V^1_p]$, i.e, for each p, $t(p)$, the translation of V^1-recognizer p, is a V^2-recognizer equivalent to p.

(b) V^1 *computably translates into* V^2 (written: $V^1 \leq V^2$) iff ($\exists$ computable t)$[t : V^1 \leq V^2]$.

(c) V^1 *limiting-computably translates into* V^2 (written: $V^1 \leq_{\lim} V^2$) iff ($\exists$ limiting-computable t)$[t : V^1 \leq V^2]$.

The next definition is also based on a definition in [Rog58].

Definition 4 (a) V is an *acceptable recognizing system* (abbreviated *a.r.s.*) iff V is a c.r.s. and ($\forall$ c.r.s. U)$[U \leq V]$.

(b) V is a *limiting-acceptable recognizing system* (abbreviated *lim-a.r.s.*) iff V is a c.r.s. and ($\forall$ c.r.s. U)$[U \leq_{\lim} V]$.

Clearly, W is an acceptable system (intuitively, a system in which one can interpret an arbitrary c.r.s.). The acceptable systems are the ones maximal with respect to $\leq$, the limiting-acceptable systems are the ones maximal with respect to $\leq_{\lim}$.

Definition 5 $V_{p,s} = W_{t(p),s}$, where t is some arbitrary but fixed computable function such that t: $V \leq W$.

Definition 6 *Friedberg computable recognizing systems* are c.r.s.'s in which there exists *exactly* one recognizer for each r.e. set.

Such systems were first shown to exist by Friedberg [Fri58], and they are useful in providing counterexamples. U and V, with or without superscripts, range over c.r.s.'s.

2.2 Learning Theory Definitions

A *sequence* σ is a mapping from an initial segment of N into $(N \cup \{\#\})$. The *content* of a sequence σ, denoted content(σ), is the set of natural numbers in the range of σ. The *length* of σ, denoted by $|\sigma|$, is the number of elements in σ. Λ denotes an empty sequence. SEQ denotes the set of all finite sequences. The set of all finite sequences of natural numbers and #'s, SEQ, can be coded onto N. This latter fact will be used implicitly in some of our proofs.

A *text* T for a language L is a mapping from N into $(N \cup \{\#\})$ such that L is the set of natural numbers in the range of T. The *content* of a text T, denoted content(T), is the set of natural numbers in the range of T. Intuitively, a text for a language is an enumeration or sequential presentation of all the objects in the language with the #'s representing pauses in the listing or presentation of such objects. For example, the only text for the empty language is just an infinite sequence of #'s. We let T, with or without superscripts, range over texts. $T[n]$ denotes the finite initial sequence of T with length n. Hence, $\text{domain}(T[n]) = \{x \mid x < n\}$.

A *language learning machine* is an algorithmic device that maps SEQ into $N \cup \{?\}$. Intuitively, the output ?'s represent the machine not yet committing to an output *program*. The reason we allow the ?'s is so that a learning machine can wait until it has seen a long enough input before it outputs its first numerical output, if at all. M ranges over language learning machines. In this paper we assume, without loss of generality, that for all $\sigma \subseteq \tau$, $[M(\sigma) \neq ?] \Rightarrow [M(\tau) \neq ?]$.

Suppose M is a learning machine and T is a text. $M(T)\downarrow$ (read: $M(T)$ *converges*) iff $(\exists i)(\stackrel{\infty}{\forall} n)\ [M(T[n]) = i]$. If $M(T)\downarrow$, then $M(T)$ is defined = the unique i such that $(\stackrel{\infty}{\forall} n)[M(T[n]) = i]$.

We now introduce a criterion for a learning machine to be considered *successful* on languages.

Definition 7 Suppose V is a c.r.s.

(a) M $\mathbf{TxtEx}_V$-*identifies* L iff ($\forall$ texts T for L)$(\exists i \mid V_i = L)[M(T)\downarrow = i]$.

(b) M $\mathbf{TxtEx}_V$-*identifies* $\mathcal{L}$, iff M $\mathbf{TxtEx}_V$-identifies each $L \in \mathcal{L}$.

(c) For all M, $\mathbf{TxtEx}_V(M) = \{L \mid M\ \mathbf{TxtEx}_V\text{-identifies } L\}$.

(d) $\mathbf{TxtEx}_V = \{\mathcal{L} \mid (\exists M)[\mathcal{L} \subseteq \mathbf{TxtEx}_V(M)]\}$.

Gold [Gol67] introduced the criterion we call $\mathbf{TxtEx}_W$.

We next introduce one-shot language identification for which the first program conjectured must be correct.

Definition 8 Suppose V is a c.r.s.

(a) M $\mathbf{TxtFin}_V$-*identifies* L iff ($\forall$ texts T for L)$(\exists i \mid V_i = L)(\exists n)[(\forall n' \geq n)[M(T[n']) = i]\ \wedge\ (\forall n' < n)[M(T[n']) = ?]]$.

(b) M $\mathbf{TxtFin}_V$-*identifies* $\mathcal{L}$, iff M $\mathbf{TxtFin}_V$-identifies each $L \in \mathcal{L}$.

(c) For all M, $\mathbf{TxtFin}_V(M) = \{L \mid M\ \mathbf{TxtFin}_V\text{-identifies } L\}$.

(d) $\mathbf{TxtFin}_V = \{\mathcal{L} \mid (\exists M)[\mathcal{L} \subseteq \mathbf{TxtFin}_V(M)]\}$.

Definition 9 Suppose V is a c.r.s.

(a) M $\mathbf{TxtMinEx}_V$-*identifies* L iff $(\forall$ texts T for $L)[M(T){\downarrow} = \mathrm{MinGram}_V(L)]$.

(b) M $\mathbf{TxtMinEx}_V$-*identifies* $\mathcal{L}$ iff M $\mathbf{TxtMinEx}$-identifies each $L \in \mathcal{L}$.

(c) For all M, $\mathbf{TxtMinEx}_V(M) = \{L \mid M$ $\mathbf{TxtMinEx}_V$-identifies $L\}$.

(d) $\mathbf{TxtMinEx}_V = \{\mathcal{L} \mid (\exists M)[\mathcal{L} \subseteq \mathbf{TxtMinEx}_V(M)]\}$.

We sometimes write $\mathbf{TxtEx}$ for $\mathbf{TxtEx}_W$ and similarly for the other criteria just discussed.

The following lemma allows us to work with a computable enumeration of learning machines.

Lemma 10 (Lemma 4.2.2B of [JORS99]) There exists a computable enumeration $M_0, M_1, \ldots$ of (total) learning machines such that, for each learning criterion $\mathcal{I}$ used in the present paper, for every $\mathcal{L} \in \mathcal{I}$, $\mathcal{L}$ is $\mathcal{I}$-identified by some machine in this enumeration. Moreover, this enumeration satisfies an S-m-n property: given a description, computable in x, of the behavior of a machine M, one can computably find a machine $M_{f(x)}$ whose $\mathcal{I}$-identification behavior is identical to that of M.

2.3 Control Structures in C.R.S.'s

[Ric80,Ric81,Roy87] show how to define control structures in the context of programming systems (effective numberings) for the partial computable functions [Rog58]. These ideas can be straightforwardly adapted to the context of c.r.s.'s. We will omit some of the details of this adaptation, but Definition 13 below will provide all that is really essential to the present paper.

Of course, **while-loop** and **if-then-else** are natural (intuitive) example control structures for systems for the partial computable functions. We exhibit in the next definition two natural example control structures in the context of c.r.s.'s. Later we present formal notions about control structures in general.

Definition 11

(a) An *instance of the control structure* **union** *in* V is a function f such that, for all p and q,

$$V_{f(p,q)} = \{x \mid x \in V_p \ \vee \ x \in V_q\}.$$

(b) An *instance of the control structure* **intersect** *in* V is a function g such that, for all p and q,

$$V_{g(p,q)} = \{x \mid x \in V_p \ \wedge \ x \in V_q\}.$$

Intuitively, for example, an instance g of **intersect** in V applied to constituent V-programs p and q, produces $g(p, q)$, a composite V-program for recognizing the intersection of the respective sets recognized by p and q.

In the present paper, it will suffice for us to consider the *extensional* [Roy87] (synonym: *denotational* [Sto77]) control structures. Instances of *extensional control structures* provide a means of forming a composite program from given constituent programs (and/or data), *where* the I/O behavior of that composite program depends only on the I/O behavior of the constituent programs (and on the data). So, for example, when applying extensional control structures, the I/O behavior of a composite program cannot generally depend on the number of symbols in or the run-time complexity of a constituent program. Clearly, in the context of c.r.s.'s, **union** and **intersect** from Definition 11 above are extensional. Also, instances of each combine *two* programs (and no data) to form a third (composite) recognizer program. [Ric80,Ric81,Roy87] provide an even more general type of control structure called *intensional* (synonym: *connotational*). Also, the extensional control structures, as rigorously defined in [Roy87], include ([Roy87, Theorem 2.3.3]) the *recursive* extensional control structures under minimal fixed point semantics.

Definition 12 [Rog67] An *enumeration operator* Θ is a mapping from 2^N to 2^N, such that for some recursively enumerable set X,

$$\text{for all } A,\ \Theta(A) = \{i \mid (\exists j)[\langle i, j\rangle \in X \ \wedge \ D_j \subseteq A\}.$$

Intuitively, an *enumeration operator* Θ is a mapping from all sets of natural numbers into the same such that some algorithm transforms arbitrary enumerations of any set A into correspondings enumerations of $\Theta(A)$. [Rog67] provides an excellent discussion of enumeration operators.

Formally, each control structure for c.r.s.'s is determined by an enumeration operator Θ. In [Ric80,Ric81,Roy87] we see that control structures in the context of programming systems for the partial computable functions are determined instead by *recursive operators* [Rog67]. As noted earlier, we provide below the definition of extensional (or denotational) control structures only since that is all that is really essential to the present paper. Also, as noted above, this definition is the obvious analog for c.r.s.'s of the corresponding concepts in [Roy87].

Definition 13

(a) Suppose $n > 0$. Suppose $0 \leq m \leq n$. Suppose Θ is an enumeration operator. Suppose V is a c.r.s.

$f : N^n \to N$ is an instance of the extensional control structure in V determined by (m,n,Θ) iff $(\forall p_1,\ldots,p_m,x_1,\ldots,x_{n-m})[V_{f(p_1,\ldots,p_m,x_1,\ldots,x_{n-m})} = \Theta(V_{p_1} \oplus \ldots \oplus V_{p_m})(x_1,\ldots,x_{n-m})]$.

(b) Suppose $n > 0$. Suppose $0 \leq m \leq n$. Suppose Θ is an enumeration operator.

The extensional control structure determined by (m,n,Θ) is $\{(V,f) \mid V$ is a c.r.s. $\wedge\ f : N^n \to N$ is an instance of the extensional control structure in V determined by $(m,n,\Theta)\}$.

(c) $\mathbf{s}$ *is an extensional control structure* iff $(\exists n > 0)(\exists m \mid 0 \leq m \leq n)(\exists$ enumeration operator $\Theta)[$ $\mathbf{s}$ is the extensional control structure determined by $(m,n,\Theta)]$.

In Definition 13(a) above $p_1,\ldots,p_m$ are program arguments, and $x_1,\ldots,x_{n-m}$ are data arguments. $f(p_1,\ldots,p_m,x_1,\ldots,x_{n-m})$ is the resultant composite V-program whose I/O behavior depends on that of the program arguments and which also depends on the data arguments. It is easy to argue that all the examples in the present paper of instances of control structures in a c.r.s. V satisfy Definition 13(a) for suitably chosen (m,n,Θ). In these examples we suppress explicit mention of the (m,n,Θ).

If f is an instance of a control structure in V, then f may or may not be computable or even limiting-computable. In the c.r.s. W, one has, of course, *computable* instances of **union** and **intersect**. Similarly, in typical, practical programming languages, one has instances of **while-loop** and **if-then-else** which are not only computable, but, since they can be realized by simple substitution of the constituent programs into some fixed template, they are computable in linear-time [Roy87,Mar89].

The learning criteria we consider in Section 3 below feature converging to a correct hypothesis *in the limit*. Hence, it is not surprising that only *limiting*-computable instances of the control structures are relevant there. However, in Section 4 further below, *computable* instances are sometimes relevant.

Case showed [Ric80,Roy87] that the acceptable programming systems (for the partial computable functions) are characterized by having a computable instance of *each* control structure. This result easily carries over to a corresponding control structure characterization of acceptable c.r.s.'s. It is a straightforward lift to show the following

Theorem 14 A c.r.s. is limiting-acceptable $\Leftrightarrow$ it has a limiting-computable instance of each *extensional* control structure.

It is currently open whether in Theorem 14 just above, the word 'extensional' can be removed. It is straightforward to show that 'extensional' can be added (before 'control structure') with no problem in the characterization of *acceptable* c.r.s.'s. These control structure characterizations of acceptability and limiting-acceptability motivate their partly *learning-theoretic* characterizations in Section 4 below.

Definition 15 We write $V \models \mathbf{s}$ to mean there is a computable instance of the control structure $\mathbf{s}$ in V, and we write $V \models \text{lim-}\mathbf{s}$ to mean that there is a limiting-computable instance of $\mathbf{s}$ in V.

We present next, examples of (extensional) control structures of relevance to the sections which follow. In the remainder of the paper, for convenience, we will many times drop the modifier 'extensional' in discussions of extensional control structures.

The first example, **s-1-1**, is a control structure intuitively for storing a datum x in a recognizing program p, more specifically, for replacing the first of two (coded) input parameters to p by the constant x. In the c.r.s. W, Kleene's S-m-n function [Rog67] essentially provides a computable instance.

Definition 16 An *instance of the control structure* **s-1-1** *in* V is a function f such that, for all p and x, $V_{f(p,x)} = \{y \mid \langle x, y\rangle \in V_p\}$.

[Rog58] characterized acceptability for programming systems (numberings) of the partial recursive functions in terms of Kleene's S-m-n Theorem. His proof straightforwardly adapts to show the following

Theorem 17

(a) For all c.r.s.'s V, V is acceptable $\Leftrightarrow V \models$ **s-1-1**.

(b) For all c.r.s.'s V, V is limiting-acceptable $\Leftrightarrow V \models$ lim-**s-1-1**.

The next example, **fin**, is a control structure which has no program arguments and one data argument x. Its instances, applied to x, return a recognizer for the canonical finite set D_x.

Definition 18 An *instance of the control structure* **fin** *in* V is a function f such that, for all x, $V_{f(x)} = D_x$.

The next example, **coinit**, is a control structure which has no program arguments and one data argument x. Its instances, applied to x, return a recognizer for the set of all integers $\geq x$.

Definition 19 An *instance of the control structure* **coinit** *in* V is a function f such that, for all x, $V_{f(x)} = \{y \mid y \geq x\}$.

The next example, **cosingle**, is a control structure which has no program arguments and one data argument x. Its instances, applied to x, return a recognizer for the set of all natural numbers $\neq x$.

Definition 20 An *instance of the control structure* **cosingle** *in* V is a function f such that, for all x, $V_{f(x)} = \{y \mid y \neq x\}$.

The next example, **proj**, is a control structure which has one program argument p and no data arguments. For **proj**, it is useful to think of V_p as a (coded) set of ordered pairs. Then an instance of **proj**, applied to p, returns a recognizer for the first (or x-axis) projection of V_p.

Definition 21 An *instance of the control structure* **proj** *in* V is a function f such that, for all p, $V_{f(p)} = \{x \mid (\exists y)[\langle x, y\rangle \in V_p]\}$.

3 Control Structure Characterizations of Learnability Results

As we noted in Section 1, in any learnability setting, hypotheses are conjectured *from some hypothesis space*. Furthermore, we noted that in the present paper we focus our attention on hypothesis spaces for recognizing the entire class of r.e. sets, and any such hypothesis space will have available some control structures but perhaps not others. The *presence* of certain control structures is, as we will see in this section, essential to certain learnability results. In the present section we first present control structure *characterizations* of some rather specific but illustrative learnability results. In the remainder of this section we consider, for the control structures involved, whether or not they are available in any hypothesis space (of the sort we consider herein). As we will see, some are always available and some are not.

In Definition 22 below, we list some standard classes in **TxtEx**.

Definition 22

(a) FiniteSets $= \{D_i \mid i \in N\}$.

(b) Co-Init $= \{L \mid (\exists i)[L = \{j \mid j \geq i\}]\}$.

(c) Co-Single $= \{L \mid (\exists i)[L = \overline{\{i\}}]\}$.

For each class from Definition 22 just above, Theorem 23 just below provides a characterization of its being in $\mathbf{TxtEx}_V$. Each such characterization features the *presence* of a particular *limiting*-computable control structure in the hypothesis space V.

Theorem 23 Suppose V is a c.r.s.. Then,

(a) FiniteSets $\in \mathbf{TxtEx}_V \Leftrightarrow V \models$ lim-**fin**.

(b) Co-Init $\in \mathbf{TxtEx}_V \Leftrightarrow V \models$ lim-**coinit**.

(c) Co-Single $\in \mathbf{TxtEx}_V \Leftrightarrow V \models$ lim-**cosingle**.

PROOF. We only prove part (b). Rest of the parts can be proved similarly. Suppose V is a c.r.s.

($\Rightarrow$): Suppose Co-Init $\in \mathbf{TxtEx}_V$ as witnessed by M. We define $f_2(\cdot,\cdot)$ as follows.

Given any i, it is possible to compute a text T_i for the language $\{n \mid n \geq i\}$ *uniformly in* i. For all i, n, let $f_2(i,n) = M(T_i[n])$. Further, let $f = \lambda i \centerdot \lim_{n\to\infty} f_2(i,n)$.

It is straightforward to show that f is an instance of lim-**coinit** in V.

($\Leftarrow$): Suppose $V \models$ lim-**coinit** as witnessed by limiting computable f. Suppose f is limiting-computable as witnessed by computable $f_2(\cdot,\cdot)$. We define M as follows.

$M(\sigma) = f_2(\min(\text{content}(\sigma) \cup \{|\sigma|\}), |\sigma|)$. It is straightforward to show that M $\mathbf{TxtEx}_V$-identifies Co-Init. ∎

We next prepare to generalize Theorem 23.

Definition 24 A class $\mathcal{L}$ of languages is *uniformly decidable* iff $\mathcal{L}$ can be written as $\{L_0, L_1, \ldots\}$, where $(\exists$ computable $d)(\forall i)[\lambda x \centerdot d(i,x) = \chi_{L_i}]$.

For example, FiniteSets is uniformly decidable: let

$$d(i,x) = \begin{cases} 1 & \text{if } x \in D_i; \\ 0 & \text{otherwise;} \end{cases}$$

then d is computable, and, with $L_i = D_i$, d witnesses that FiniteSets is a uniformly decidable class. Similarly, we see that Co-Init and Co-Single are also uniformly decidable classes. Of course all these classes are in **TxtEx**. Actually, uniformly decidable classes of languages are ubiquitous in computational learning theory [ZL95] and are many times also called *indexed families of recursive languages*. Important further examples of such classes are the class of all pattern languages [Ang80b,Ang80a] and the class of all context free languages [HU79]. The former is in **TxtEx** [Ang80b,Ang80a], but the latter is not [Gol67].

Next, we define a class of control structures useful for uniformly decidable classes. Just after that we provide Theorem 26 which generalizes Theorem 23 above.

Let $\mathcal{L} = \{L_i \mid i \in N\}$ be a uniformly decidable class of recursive languages, where $(\exists$ computable $d)(\forall i)[\lambda x \,\boldsymbol{.}\, d(i,x) = \chi_{L_i}]$. We associate with $\mathcal{L}$ (and the listing of $\mathcal{L}$ as $L_0, L_1, \ldots$) a control structure $\mathbf{cs}_{\mathcal{L}}$ which has no program arguments and one data argument i. An instance of $\mathbf{cs}_{\mathcal{L}}$, applied to i, returns a recognizer for the language L_i. N.B. The parameter i here *within the system* V does serve as a datum; however, within the subrecursive system $\langle L_i \mid i \in N\rangle$ it can be construed as a program (for deciding L_i).

Definition 25 For $\mathcal{L}$ as just above, an *instance of the control structure* $\mathbf{cs}_{\mathcal{L}}$ *in* V is a function f such that, for all i,

$$V_{f(i)} = L_i.$$

For example, if $\mathcal{L} =$ FiniteSets and $L_i = D_i$, then $\mathbf{cs}_{\mathcal{L}} = \mathbf{fin}$.

Theorem 23 generalizes as follows.

Theorem 26 Suppose $\mathcal{L} \in \mathbf{TxtEx}$ is a uniformly decidable class. Then, $(\forall V)[\mathcal{L} \in \mathbf{TxtEx}_V \Leftrightarrow V \models \text{lim-}\mathbf{cs}_{\mathcal{L}}]$.

In the remainder of this section we present (among other things) results showing that some of the necessary control structures featured above in this section are present in every c.r.s. and some are not.

From Theorem 27 and Theorem 29 below, we will see that FiniteSets can be **TxtEx**-identified in all c.r.s.'s, but that there is a c.r.s. in which Co-Init cannot be **TxtEx**-identified. From this perspective, then, FiniteSets is *easier* than Co-Init. By contrast, with respect to an *intrinsic complexity* notion from [FKS95,JS96], FiniteSets is *harder* than Co-Init for **TxtEx**-identification.

Theorem 27 For all c.r.s.'s V, FiniteSets $\in \mathbf{TxtEx}_V$.

Proof. Suppose V is a c.r.s. We define a machine M such that M $\mathbf{TxtEx}_V$-identifies FiniteSets.

Let $M(\sigma) = i$, where i is the least $j \le |\sigma| = n$ such that $V_{j,n} = \text{content}(\sigma)$, if any; 0, otherwise.

Clearly, given a text for a finite set, M converges in the limit to the minimum recognizer for that set. ∎

From Theorem 23 and Theorem 27, we have the following

Corollary 28 For all c.r.s.'s V, $V \models$ lim-**fin**.

For the class Co-Init, however, we get the following result. Its proof is technically interesting since it involves a pleasing, subtle non-constructivity in the way the entire class of r.e. sets is embedded in the example c.r.s. of the Theorem.

Theorem 29 There exists a c.r.s. V such that $V \not\models$ lim-**coinit** (and hence Co-Init $\notin \mathbf{TxtEx}_V$).

PROOF. We use the symbol $\downarrow$ to denote that a computation halts. We define V in stages below. Go to stage 0.

Begin stage n

For all $i \leq n$, do the following steps. For all i, let $s_i = \max(\{s \leq n \mid \varphi_{i,n}(i,s)\downarrow\})$; let $p_i = \varphi_i(i, s_i)$, if $s_i \neq 0$ (recall that $\max(\emptyset) = 0$); otherwise, p_i is undefined.

1. For all $q \in \{p_i \mid i \leq n\}$, let $\text{ClaimSet}_q = \{i \leq n \mid p_i = q\}$; for all other q, let $\text{ClaimSet}_q = \emptyset$.
2. If $\min(W_{i,n}) \notin \text{ClaimSet}_{2i}$, then enumerate $W_{i,n}$ into V_{2i}.
3. If $\min(W_{i,n}) \notin \text{ClaimSet}_{2i+1}$, then enumerate $W_{i,n}$ into V_{2i+1}.
4. Go to stage $n + 1$.

End stage n.

Claim 30 $V \not\models$ lim-**coinit** (and hence Co-Init $\notin \mathbf{TxtEx}_V$).

PROOF OF CLAIM 30. It suffices to show that there is no limiting-computable function f such that, for all i, $V_{f(i)} = \{x \mid x \geq i\}$. Suppose f is limiting-computable as witnessed by computable $f_2(\cdot,\cdot)$. Let $\varphi_i = f_2$. $\varphi_i(i,\infty)\downarrow = p$, say. Therefore, $V_p = \{x \mid x \geq i\}$. Hence, from the construction, $V_p = W_q$, where $q = \lfloor p/2 \rfloor$. Thus, $i = \min(W_q)$. Therefore, $(\overset{\infty}{\forall} n)[i \in W_{q,n}]$. Also, $(\overset{\infty}{\forall} n)[i \in \text{ClaimSet}_p$ at stage $n]$. Hence, by the construction above, V_p is a finite set, a contradiction. □ (Claim 30)

Claim 31 For all p, there exists an i such that $V_i = W_p$.

PROOF OF CLAIM 31. Let $j = \min(W_p)$. From the construction, at any stage s, either $j \notin \text{ClaimSet}_{2p}$ or $j \notin \text{ClaimSet}_{2p+1}$. Hence, at least one of V_{2p} and V_{2p+1} is the set W_p. □ (Claim 31)

It follows from the above claims that V is a c.r.s., $V \not\models$ lim-**coinit**, and Co-Init $\notin \mathbf{TxtEx}_V$. ▮ (Theorem 29)

The proof of Theorem 29, can be easily generalized to uniformly decidable classes of *infinite* recursive languages to give

Theorem 32 Suppose $\mathcal{L} \in \mathbf{TxtEx}$ is a *infinite* uniformly decidable class containing only *infinite* (recursive) languages. Then, $(\exists$ a c.r.s. $V)[\mathcal{L} \notin \mathbf{TxtEx}_V]$.

We then have the following

Corollary 33 There exists a c.r.s. V such that $V \not\models$ lim-**cosingle** (and Co-Single $\notin \mathbf{TxtEx}_V$).

In another vein, Theorem 27 gives us the following

Corollary 34 $(\exists \mathcal{L} \mid \text{card}(\mathcal{L})$ infinite$)(\forall$ c.r.s. $V)[\mathcal{L} \in \mathbf{TxtEx}_V]$.

The immediately above corollary contrasts with [FKS94, Lemmas 25 & 26] which yield programming systems (for the partial computable functions) with respect to which one cannot learn in the limit any infinite class of (total) computable functions. An explanation for this and the next contrasting result is that, in learning computable functions, there are no finite objects to be learned.

As an even more contrasting result, an easy generalization of the proof of Theorem 27 gives,

Theorem 35 Suppose $\mathcal{L} \in \mathbf{TxtEx}$ contains at most *finitely* many *infinite* sets (with no restriction on how many finite sets it contains). Then $(\forall$ c.r.s. $V)[\mathcal{L} \in \mathbf{TxtEx}_V]$.

From Theorems 26 and 35 we have, then, the following

Corollary 36 Suppose uniformly decidable $\mathcal{L} = \{L_0, L_1, \ldots\} \in \mathbf{TxtEx}$ contains at most *finitely* many *infinite* sets, where $(\exists$ computable $d)(\forall i)[\lambda x \centerdot d(i,x) = \chi_{L_i}]$. Then $(\forall$ c.r.s. $V)[V \models$ lim-$\mathbf{cs}_{\mathcal{L}}]$

4 Partly Learning-Theoretic Characterizations of Having "All" Control Structures

In this section we present our two main characterization theorems, Theorems 39 and 40. The first characterizes $\mathbf{TxtFin}_V$ being $=$ $\mathbf{TxtFin}$ *and* the presence of a computable instance of **proj**, in V as: V has computable instances of all (extensional) control structures. The second characterizes $\mathbf{TxtEx}_V$ being $= \mathbf{TxtEx}$ *and* the presence of a limiting-computable instance of **proj**, in V as: V has limiting-computable instances of all extensional control structures. Of course, by remarks in Section 2.3 above, these are just characterizations of acceptability and limiting-acceptability, respectively; hence, we express them in such terms. In a sense, then, **proj** epitomizes the control structures whose *presence* or *absence* is *not* relevant for invariance of the learning classes. As we will see, the hardest part of each of these theorems is its crucial *furthermore* clause. After our main theorems we consider a number of related matters and consequences.

The following Theorem is useful in proving part of our first main theorem and is of interest in its own right. Essentially, it implies that there is a c.r.s. V which: has limiting-computable instances of all extensional control structures, is missing any *computable* instance of some extensional control structure, but, nonetheless, the missing computable instance does not lessen learning power. By Theorem 17, V has a limiting-computable instance of the extensional control structure **s-1-1**, but V does not have a computable instance of **s-1-1**.

Efim Kinber suggested the c.r.s. used in the particular proof we give of this theorem.

Theorem 37 There exists a limiting-acceptable c.r.s. V that is not acceptable, such that $\mathbf{TxtEx}_V = \mathbf{TxtEx}$ and $\mathbf{TxtFin}_V = \mathbf{TxtFin}$.

PROOF. We define a c.r.s. V as follows.

$$V_i = \begin{cases} \{0\} & \text{if } i = 0; \\ \emptyset & \text{if } i > 0 \text{ and } W_i = \{0\}; \\ W_i & \text{otherwise.} \end{cases}$$

Clearly, V is a c.r.s. Also, it is straightforward to see that $\mathbf{TxtEx}_V = \mathbf{TxtEx}$, $\mathbf{TxtFin}_V = \mathbf{TxtFin}$ and that V is limiting-acceptable. Suppose by way of contradiction that V is an a.r.s. Then, from the definition of a.r.s. (Definition 4), $W \leq V$. Suppose t computable such that $t : W \leq V$. Then, $W_i = \{0\} \Leftrightarrow t(i) = 0$, and, hence, $\{i \mid W_i = \{0\}\}$ is recursive, a contradiction to Rice's Theorem [Rog67,DSW94]. ∎

By replacing $\{0\}$ in the above proof to $\{j\}$, for arbitrary $j \in N$, we obtain

Theorem 38 There exists infinitely many pairwise $\leq$-incomparable non-acceptable c.r.s.'s $V_1, V_2, \ldots$, such that $\mathbf{TxtEx}_{V_1} = \mathbf{TxtEx}_{V_2} = \ldots = \mathbf{TxtEx}$.

Here is our first main

Theorem 39 V is acceptable $\Leftrightarrow$ [$\mathbf{TxtFin}_V = \mathbf{TxtFin} \ \wedge \ V \models \mathbf{proj}$].
Furthermore, the clauses in the right hand side are independent of each other.

The proof of the $\Leftrightarrow$ part of Theorem 39 is a straightforward variant of the proof of the $\Leftrightarrow$ part of Theorem 40 below. For the furthermore part: $(\exists V)[V \models \mathbf{proj} \not\Rightarrow \mathbf{TxtFin}_V = \mathbf{TxtFin}]$ follows from the $\Leftrightarrow$ part of Theorem 39 and Theorem 41 below; $(\exists V)[\mathbf{TxtFin}_V = \mathbf{TxtFin} \not\Rightarrow V \models \mathbf{proj}]$ follows from the $\Leftrightarrow$ part of Theorem 39 and Theorem 37 above.

Our second main theorem is next.

Theorem 40 V is limiting-acceptable $\Leftrightarrow$ [$\mathbf{TxtEx}_V = \mathbf{TxtEx} \ \wedge \ V \models$ lim-$\mathbf{proj}$].
Furthermore, the clauses in the right hand side are independent of each other.

We first prove the $\Leftrightarrow$ part of this theorem. Then the furthermore part will follow from this $\Leftrightarrow$ part together with Theorems 41 and 42 below.

PROOF OF THE $\Leftrightarrow$ PART OF THEOREM 40.
($\Rightarrow$): Suppose V is limiting-acceptable. Let $f : W \leq_{\mathrm{lim}} V$. Let f be limiting-computable as witnessed by computable $f_2(\cdot,\cdot)$ (i.e., $f = \lambda x \,.\, \lim_{n\to\infty} f_2(x,n)$).

Suppose $\mathcal{L} \in \mathbf{TxtEx}_W$ as witnessed by M. We show that $\mathcal{L} \in \mathbf{TxtEx}_V$.

We define a learning machine M' thus: For all texts T, for all n, let $M'(T[n]) = f_2(M(T[n]), n)$. Clearly, for all T, $M(T)\!\downarrow = p \Rightarrow M'(T)\!\downarrow = f(p)$. Hence, M' $\mathbf{TxtEx}_V$-identifies $\mathcal{L}$.

Also, clearly, $W \models \mathbf{proj}$. Let g be a computable instance of $\mathbf{proj}$ in W. So, for all i, $W_{g(i)} = \{x \mid (\exists y)[\langle x, y\rangle \in W_i]\}$.

We next define computable $h_2(\cdot,\cdot)$ such that $h = \lambda i \,.\, \lim_{n\to\infty} h_2(i,n)$ is a limiting-computable instance of $\mathbf{proj}$ in V. Let t be a computable function such that $t : V \leq W$. Clearly, such a t exists.

Let $h_2(i,n) = f_2(g(t(i)), n)$. For all i, $h(i) = \lim_{n\to\infty} h_2(i,n) = \lim_{n\to\infty} f_2(g(t(i)), n) = f(g(t(i)))$. Therefore, $V_{h(i)} = V_{f(g(t(i)))} = W_{g(t(i))}$. But $W_{g(t(i))} = \{x \mid (\exists y)[\langle x, y\rangle \in W_{t(i)}]\}$. Since $W_{t(i)} = V_i$, h is a limiting-computable instance of $\mathbf{proj}$ in V. So, $V \models$ lim-$\mathbf{proj}$.

($\Leftarrow$): Suppose $\mathbf{TxtEx}_V = \mathbf{TxtEx}_W$ and h is a limiting-computable instance of $\mathbf{proj}$ in V (as witnessed by computable $h_2(\cdot,\cdot)$).

We use the class $\mathcal{L}_{\mathbf{TxtEx}}$ from [JS96], which we describe below. This class $\mathcal{L}_{\mathbf{TxtEx}}$ is shown to be in $\mathbf{TxtEx}_W$ in [JS96].

Let $S_L^j = \{\langle x, j\rangle \mid x \in L\}$. Then, $\mathcal{L}_{\mathbf{TxtEx}} = \{S_L^j \mid L \in \mathbf{TxtEx}(M_j)\}$.

Let M be a learning machine that $\mathbf{TxtEx}_V$-identifies $\mathcal{L}_{\mathbf{TxtEx}}$. Also, there exists a computable f such that for each i, $M_{f(i)}\mathbf{TxtEx}_W$-identifies W_i. Hence, for all i, the language $S_{W_i}^{f(i)}$ in $\mathcal{L}_{\mathbf{TxtEx}}$ is $\mathbf{TxtEx}_V$-identified by M.

For each language, $S_{W_i}^{f(i)}$, let T_i be a text for this language, that can be *computed uniformly in i.*

We next define computable t_2 such that $t = \lambda i \centerdot \lim_{n\to\infty} t_2(i,n)$ is such that $t : W \leq_{\lim} V$.

Let $t_2(i,n) = h_2(M(T_i[n]), n)$.

Now, for each i, $t(i) = \lim_{n\to\infty} t_2(i,n) = \lim_{n\to\infty} h_2(M(T_i[n]), n) = \lim_{n\to\infty} h_2(M(T_i), n) = h(M(T_i))$.

Hence, $V_{t(i)} = V_{h(M(T_i))} = W_i$.

Therefore, $W \leq_{\lim} V$, i.e., V is limiting-acceptable. ▮ ($\Leftrightarrow$ part of Theorem 40)

Theorem 41 $(\exists V)[V \models \mathbf{proj}$ and $\mathbf{TxtFin} \not\subseteq \mathbf{TxtEx}_V]$.

(Thus, the above V is not limiting-acceptable and $V \models$ lim-$\mathbf{proj}$.)

PROOF. Let $L_j = \{\langle j, x\rangle \mid x \in N\}$. Let $\mathcal{L} = \{L_j \mid j \in N\}$. Clearly, $\mathcal{L} \in \mathbf{TxtFin}$. We will construct a c.r.s. V such that $V \models \mathbf{proj}$ but $\mathcal{L} \notin \mathbf{TxtEx}_V$. This will prove the theorem.

For a language L, let $\mathrm{Proj}(L) = \{x \mid (\exists y)[\langle x, y\rangle \in L]\}$. Let $\mathrm{Proj}^0(L)$ denote L, and $\mathrm{Proj}^{i+1}(L)$ denote $\mathrm{Proj}(\mathrm{Proj}^i(L))$.

Let $\mathrm{ProjSet}(L) = \{L' \mid (\exists i)[L' = \mathrm{Proj}^i(L)]\}$.

Without loss of generality assume that W is an a.r.s. such that, for all j, which are not powers of 2, $W_j = L_j$. Note that this implies, there exist infinitely many j such that (1) $\mathrm{MinGram}(L_j) = j$ and (2) $(\forall i < j)[L_j \notin \mathrm{ProjSet}(W_i)]$. This is what we will utilize in our construction.

We now define V. For all x, y, let $V_{\langle x, y+1\rangle} = \mathrm{Proj}(V_{\langle x,y\rangle})$. (Note that this ensures $V \models \mathbf{proj}$.)

We now only need to define $V_{\langle x,0\rangle}$ for each x. We do so next. Let h be defined as follows:

For all i, for all $j \leq (i+1)^2$, $h(i,j) = \Sigma_{k=0}^{i-1}((k+1)^2+1)+j$.

Note that $S_i = \{h(i,j) \mid j \leq (i+1)^2\}$ is a disjoint partition of N and $\mathrm{card}(S_i) = (i+1)^2+1$.

We define, for each i and $j \leq (i+1)^2$, $V_{\langle h(i,j),0\rangle}$, as follows. $V_{\langle h(i,j),0\rangle}$ will either be W_i or a finite subset of W_i.

Let T_i denote some standard (effective in i) text for W_i.

Definition of $V_{\langle h(i,j),0\rangle}$.
Go to stage 0.
Stage s.
1. If $\{\langle h(i',j),k\rangle \mid i',k \in N\} \cap \{M_v(T_w[s]) \mid v \leq i \ \wedge \ w \leq i\} = \emptyset$, then
   Enumerate $W_{i,s}$.
2. Go to stage $s+1$.

End stage s

It is straightforward to verify that $V_{\langle h(i,j),0\rangle} = W_i$, or a finite subset of W_i. Also, for each i, there exists a j such that $V_{\langle h(i,j),k\rangle} = W_i$. Thus V is a c.r.s. Moreover, if M_v on W_w converges to $\langle h(i',j'),k'\rangle$, then, for all $i \geq \max(\{v,w\})$, $V_{\langle h(i,j'),0\rangle}$ is finite.

Suppose by way of contradiction that M_v $\mathbf{TxtEx}_V$-identifies $\mathcal{L}$. Let i be large enough such that $i > v$, $\mathrm{MinGram}_W(L_i) = i$, $(\forall i' < i)[L_i \notin \mathrm{ProjSet}(W_{i'})]$. Note that there exists such an i, by the assumption on W. We claim that M_v cannot $\mathbf{TxtEx}_V$-identify L_i.

So suppose $M_v(L_i)\downarrow = \langle h(i',j'),k'\rangle$. If $i' \geq i$, then by construction above $V_{\langle h(i',j'),0\rangle}$ is finite. Thus $V_{\langle h(i',j'),k'\rangle} \neq L_i$. If $i' < i$, then either $V_{\langle h(i',j'),k'\rangle}$ is finite or a member of $\mathrm{ProjSet}(W_{i'})$. But $L_i \notin \mathrm{ProjSet}(W_{i'})$. Thus $V_{\langle h(i',j'),k'\rangle} \neq L_i$.

It follows that $\mathcal{L} \notin \mathbf{TxtEx}_V$. Thus, by the $\Leftrightarrow$ part of Theorem 40, we have that V is not limiting-acceptable. ∎

Theorem 42 $(\exists V)[V$ is not limiting-acceptable and $\mathbf{TxtEx}_V = \mathbf{TxtEx}]$.

The proof of this theorem proceeds employing a series of lemmas and propositions.

Let $\mathrm{Init} = \{L \mid (\exists j)[L = \{i \mid i < j\}]\}$.

Lemma 43 Suppose V is a c.r.s. such that $V_0 = N$. Then one can effectively (in algorithmic description of V) obtain a Friedberg c.r.s. U and a limiting recursive function f such that,

$(\forall i \mid V_i \notin (\{N\} \cup \text{Init}) \ \wedge \ i = \text{MinGram}_V(V_i))[U_{f(i)} = V_i]$.

PROOF. Odifreddi's construction ([Odi89, Theorem II.5.22, Page 230]) proves this lemma. ▮ (Lemma 43)

Proposition 44 Suppose $\mathcal{L}'$ is finite, U is a c.r.s., $\mathcal{L} \cup \mathcal{L}' \in \mathbf{TxtEx}$, and $\mathcal{L} \in \mathbf{TxtEx}_U$. Then $\mathcal{L} \cup \mathcal{L}' \in \mathbf{TxtEx}_U$.

PROOF OF PROPOSITION 44. Suppose the hypothesis. Let M be such that $\mathcal{L} \cup \mathcal{L}' \subseteq \mathbf{TxtEx}(M)$. Let M' be such that $\mathcal{L} \subseteq \mathbf{TxtEx}_U(M')$. Let $S = \{M(L) \mid L \in \mathcal{L}'\}$. For each $i \in S$ and $L \in \mathcal{L}'$, such that $M(L) = i$, let j_i be such that $U_{j_i} = L$.

Define M'' as follows:

$$M''(\sigma) = \begin{cases} j_{M(\sigma)}, & \text{if } M(\sigma) \in S; \\ M'(\sigma), & \text{otherwise.} \end{cases}$$

It is straightforward to verify that M'' $\mathbf{TxtEx}_U$-identifies, $\mathcal{L} \cup \mathcal{L}'$. ▮ (Proposition 44)

As a corollary to Theorem 27 and its proof we have,

Corollary 45 For all c.r.s.'s U, Init $\in \mathbf{TxtMinEx}_U$.

The following Lemma is proved using Lemma 43, Proposition 44, and Corollary 45.

Lemma 46 Suppose V is a c.r.s. Then one can effectively (in algorithmic description of V) construct a Friedberg c.r.s. U such that $\mathbf{TxtMinEx}_V \subseteq \mathbf{TxtEx}_U = \mathbf{TxtMinEx}_U$.

PROOF OF LEMMA 46. Without loss of generality assume that $V_0 = N$. We assume this property of V just for ease of notation, since one can effectively transform V into a c.r.s. V' such that $V'_0 = N$ and $\mathbf{TxtMinEx}_V \subseteq \mathbf{TxtMinEx}_{V'}$ (to do this, let $V'_0 = N$, and $V'_{i+1} = V_i$).

Let U be the Friedberg c.r.s. which we get by using Lemma 43.

Suppose $\mathcal{L} \in \mathbf{TxtMinEx}_V$. Clearly, $\mathcal{L} - (\{N\} \cup \text{Init}) \in \mathbf{TxtEx}_U$ (since, using f as in Lemma 43, we can convert, in the limit, minimal V-recognizer for $L \notin (\{N\} \cup \text{Init})$, to U-recognizer for L). Let $\mathcal{L}' = \mathcal{L} \cap (\{N\} \cup \text{Init})$.

We now consider two cases:

Case 1: $N \in \mathcal{L}$.

In this case, clearly, $\mathcal{L}'$ must be finite. Hence we get $\mathcal{L} \in \mathbf{TxtEx}_U$ by Proposition 44.

Case 2: $N \notin \mathcal{L}$.

In this case, clearly $\mathcal{L}' \subseteq \text{Init}$. Let M be a machine which witnesses that $\mathcal{L} - (\{N\} \cup \text{Init}) \in \mathbf{TxtEx}_U$. Let M' be a machine which witnesses that $\text{Init} \in \mathbf{TxtEx}_U$.

Define M'' as follows:

$$M''(\sigma) = \begin{cases} M'(\sigma), & \text{if content}(\sigma) \in \text{Init}; \\ M(\sigma), & \text{otherwise.} \end{cases}$$

It is straightforward to verify that M'' $\mathbf{TxtEx}_U$-identifies $\mathcal{L}$. ∎ (Lemma 46)

For proving Lemma 49 below, we need the notion of order independence.

Definition 47 ([BB75,Ful90]) A machine M is *order independent* iff, for all texts T and T', if $\text{content}(T) = \text{content}(T')$ and $M(T)\downarrow$, then $M(T')\downarrow = M(T)$.

Lemma 48 ([BB75,Ful90]) Suppose M is given. Then one can effectively (from M) construct an order independent machine M' such that, for all c.r.s.'s V, $\mathbf{TxtEx}_V(M) \subseteq \mathbf{TxtEx}_V(M')$.

For an order independent machine M we often use $M(L)$ to denote $M(T)$, for any text T for L. Note that this notion of $M(L)$ is well defined for order independent machines M.

Lemma 49 Suppose M is given. Let $\mathcal{L} = \mathbf{TxtEx}(M)$. Then one can effectively construct a c.r.s. V and a machine M' such that
(a) $\mathcal{L} \subseteq \mathbf{TxtEx}_V(M')$, and
(b) For infinite $L \in \mathcal{L}$, M' $\mathbf{TxtMinEx}_V$-identifies L.

Proof of Lemma 49. By Lemma 48 one can, effectively from M, construct an order independent machine M' such that $\mathbf{TxtEx}(M) \subseteq \mathbf{TxtEx}(M')$.

Let T_j denote a text for W_j, which can be obtained effectively from j. V_i is defined in stages as follows.

Definition of V_i.
Go to stage 0.

Stage s.
1. If $[M'(T_i[s]) \neq M'(T_i[s+1])]$ or $[M'(T_i[s]) \leq i]$, then
   Enumerate $W_{i,s}$.
2. Go to stage $s+1$.
End stage s

We now prove that V satisfies the requirements of the theorem.

V is a c.r.s.: Consider any language L. If $M'(L)\uparrow$, then clearly $W_i = L \Rightarrow V_i = L$. If $M'(L)\downarrow = j$, then for any $i > j$, such that $W_i = L$, we have $V_i = L$. Thus V is a c.r.s.

M' $\mathbf{TxtEx}_V$-identifies $\mathcal{L}$: For $L \in \mathcal{L}$, $M'(L)\downarrow = j$, such that $L = W_j$. Thus $M'(T_j) = j$. Thus, by construction $V_j = W_j = L$.

For infinite $L \in \mathcal{L}$, M' $\mathbf{TxtMinEx}_V$-identifies L: First note that, for all j, either V_j is finite or $V_j = W_j$. Thus it suffices to show that for every infinite $L \in \mathcal{L}$, for all i such that $W_i = L$ and $i < M'(L)$, V_i is finite. But this immediately follows from the construction, since the if condition (in construction of V_i) holds only for finitely many stages. ▮ (Lemma 49)

Lemma 50 Suppose V is a c.r.s. Further suppose M and $\mathcal{L}$ are such that
(a) M $\mathbf{TxtEx}_V$-identifies $\mathcal{L}$, and
(b) For all infinite $L \in \mathcal{L}$, M $\mathbf{TxtMinEx}_V$-identifies L.
Then, $\mathcal{L} \in \mathbf{TxtMinEx}_V$.

Proof of Lemma 50. Suppose the hypothesis. We construct M' which $\mathbf{TxtMinEx}_V$-identifies $\mathcal{L}$. Let M' be defined as follows:

$M'(T[n]) = \min(\{k \leq n \mid V_{M(T[n]),n} = V_{k,n}\})$

Now suppose T is a text for $L \in \mathcal{L}$.

If L is infinite, then, for all but finitely many n, $M(T[n]) = \mathrm{MinGram}_V(L)$. Thus, for all but finitely many n, $M'(T[n]) = \mathrm{MinGram}_V(L)$.

If L is finite, then, for all but finitely many n, $V_{M(T[n]),n} = L$. Thus, for all but finitely many n, $\min(\{k \leq n \mid V_{M(T[n]),n} = V_{k,n}\}) = \mathrm{MinGram}_V(L)$. Thus $M'(T) = \mathrm{MinGram}_V(L)$.

This proves that $\mathcal{L} \in \mathbf{TxtMinEx}_V$. ▮ (Lemma 50)

We get the following corollary from Lemmas 49 and 50.

Corollary 51 For any inductive inference machine M, one can effectively (in M) construct a c.r.s. V such that $\mathbf{TxtEx}(M) \in \mathbf{TxtMinEx}_V$.

As a corollary to Corollary 51 and Lemma 46 we get

Corollary 52 For any inductive inference machine M, one can effectively (in M) construct a Friedberg c.r.s U such that $\mathbf{TxtEx}(M) \in \mathbf{TxtEx}_U$.

A sequence of c.r.s.'s $V^0, V^1, \ldots$ is an *an r.e. sequence of c.r.s.'s* just in case the set $\{\langle\langle i,j\rangle, x\rangle \mid x \in V^i_j\}$ is recursively enumerable. The *direct sum* of an r.e. sequence of c.r.s.'s, $V^0, V^1, \ldots$ is defined to be the c.r.s. V such that for all i, j, $V_{\langle i,j\rangle} = V^i_j$.

Finally, by an straightforward modification of the proof of the main theorem in [Kum89], we get the following

Lemma 53 The direct sum of an r.e. sequence of Friedberg c.r.s.'s is *never* limiting-acceptable.

PROOF OF THEOREM 42. For each i, let U^i be a Friedberg c.r.s. obtained effectively from i such that $\mathbf{TxtEx}(M_i) \in \mathbf{TxtEx}_{U^i}$ (the effectiveness is from Corollary 52). Let U be a direct sum of $U^0, U^1, \ldots$. It follows that $\mathbf{TxtEx} = \mathbf{TxtEx}_U$. Also, by Lemma 53, U is not limiting-acceptable. The theorem follows. ▮ (Theorem 42)

We earlier showed the $\Leftrightarrow$ part of Theorem 40. This together with Theorems 41 and 42 give us the furthermore part of Theorem 40. ▮ (Theorem 40)

It is straightforward to show that a single Friedberg c.r.s. is not limiting-acceptable, yet Theorem 54 just below implies no single Friedberg c.r.s. can witness the truth of Theorem 42 above. Theorem 54 is a consequence of a straightforward modification of the proof of Theorem 4 from [FKW82].

Theorem 54 For all Friedberg c.r.s.'s U, $\mathbf{TxtEx}_U \subset \mathbf{TxtEx}$.

The next result shows us that a c.r.s. V is limiting-acceptable just in case one can computably (or equivalently, limiting-computably) translate **TxtEx**-identifying machines to $\mathbf{TxtEx}_V$-identifying machines.

Theorem 55 The following three clauses are equivalent

(1) V is limiting–acceptable
(2) $(\exists \text{ computable } g)(\forall M)[\mathbf{TxtEx}(M) \subseteq \mathbf{TxtEx}_V(g(M))]$
(3) $(\exists \text{ limiting-computable } h)(\forall M)[\mathbf{TxtEx}(M) \subseteq \mathbf{TxtEx}_V(h(M))]$.

PROOF. ((1) $\Rightarrow$ (2)): Let t be a limiting computable translator as witnessed by $t_2(\cdot,\cdot)$ from W to V. For all M, define $g(M)$ as follows. For all T, n, let $g(M)(T[n]) = t_2(M(T[n]), n)$. Clearly, $\mathbf{TxtEx}(M) \subseteq \mathbf{TxtEx}_V(g(M))$.

((2) $\Rightarrow$ (3)): Follows easily.

((3) $\Rightarrow$ (1)): Suppose h is limiting-computable as in the hypothesis. Suppose h_2 witness that h is limiting-computable. There exists a computable f such that, for all i, for all σ, a finite initial segment of a text, $M_{f(i)}(\sigma) = i$. Therefore, $W_i \in \mathbf{TxtEx}_W(M_{f(i)})$. Hence $W_i \in \mathbf{TxtEx}_V(h(M_{f(i)}))$.

Given any recognizer i, it is possible to computably generate a text T_i for the language W_i uniformly in i. V is limiting-acceptable as witnessed by the limiting-computable translator t below.

Let $t_2(i,n) = h_2(M_{f(i)}, n)(T_i[n])$. Let $t(i) = \lim_{n\to\infty} t_2(i,n)$. Then, for all i, $V_{t(i)} = V_{h(M_{f(i)})(T_i)} = W_i$. ∎

Using Theorem 55 and Theorem 42, we get

Corollary 56 $(\exists V)[\mathbf{TxtEx}_V = \mathbf{TxtEx} \not\Rightarrow (\exists$ computable $g)(\forall M)[\mathbf{TxtEx}(M) \subseteq \mathbf{TxtEx}_V(g(M))]]$.

5 Conclusions, Problems, and Future Directions

Theorem 23 and its generalization, Theorem 26, present control structures whose *presence* is *needed* for full learning power. Some of these necessary control structures are present in any c.r.s. (Corollaries 28 and 36). That some are not follows from Theorems 23, 26, 29 and 32 and Corollary 33.

Theorem 17 together with Theorems 37 and 42 show that the presence of neither a computable instance of **s-1-1** nor a limiting-computable one is needed for full learning power.

By Theorem 41, there is a c.r.s. V where a computable instance of the control structure **proj** *is* available, but learning in the limit with V as the hypothesis space is, nonetheless, extremely weakened. The main theorems (Theorems 39 and 40) more generally indicate that **proj** *epitomizes* the control structures whose *presence* needn't help and whose *absence* needn't hinder learning power. We do not yet know how to otherwise insightfully *characterize* the control structures similarly irrelevant for learning class invariance.

It would be interesting to get learnability results about control structures in *sub*recursive hypothesis spaces [Roy87,ZL95,RC94]. Subrecursive systems have no analog of acceptability [Roy87]; however, back in the general recursive setting, it would be nice to investigate whether there exist pure learning-theoretic results completely characterizing each of acceptability and limiting-acceptability .

What we originally set out to do (for the principal learning criteria of this paper) was to

(1) find a set of control structures $\mathbf{S}$ such that $\mathbf{TxtFin}_V = \mathbf{TxtFin} \Leftrightarrow (\forall \mathbf{s} \in \mathbf{S})[V \models \mathbf{s}]$; and
(2) find a set of control structures $\mathbf{S}$ such that $\mathbf{TxtEx}_V = \mathbf{TxtEx} \Leftrightarrow (\forall \mathbf{s} \in \mathbf{S})[V \models \text{lim-}\mathbf{s}]$.

This remains to be done.

Acknowledgements

We would like to thank Ganesh Baliga for helpful discussions, encouragement, and for suggesting some lines of research that, in part, led to the present paper. We would also like to thank each of Efim Kinber and Rolf Wiehagen for helpful discussions and examples. Sanjay Jain was supported in part by NUS grant number RP3992710.

References

[All78] J. Allen. *Anatomy of Lisp.* McGraw-Hill, New York, NY, 1978.

[Ang80a] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[Ang80b] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

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

[Blu67] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14:322–336, 1967.

[BS94] G. Baliga and A. Shende. Learning-theoretic perspectives of acceptable numberings. In *Third International Symposium on Artificial Intelligence and Mathematics*, 1994.

[Chu41] A. Church. *The Calculi of Lambda Conversion.* Princeton Univ. Press, 1941.

[CSM92] D. Kedzier C. Sammut, S. Hurst and D. Michie. Learning to fly. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Conference on Machine Learning.* Morgan Kaufmann, 1992.

[DSW94] M. Davis, R. Sigal, and E. Weyuker. *Computability, Complexity, and Languages.* Academic Press, second edition, 1994.

[FKS94] R. Freivalds, M. Karpinski, and C. Smith. Co-learning of total recursive functions. In *Proceedings of the Seventh Annual Conference on Computational Learning Theory*, pages 190–197. ACM Press, 1994.

[FKS95] R. Freivalds, E. Kinber, and C. Smith. On the intrinsic complexity of learning. *Information and Computation*, 123(1):64–71, 1995.

[FKW82] R. Freivalds, E. Kinber, and R. Wiehagen. Inductive inference and computable one-one numberings. *Zeitschr. j. math. Logik und Grundlagen d. Math. Bd.*, 28:463–479, 1982.

[FKW84] R. Freivalds, E. Kinber, and R. Wiehagen. Connections between identifying functionals, standardizing operations, and computable numberings. *Zeitschr. j. math. Logik und Grundlagen d. Math. Bd.*, 30:145–164, 1984.

[Fri58] R. Friedberg. Three theorems on recursive enumeration. *Journal of Symbolic Logic*, 23(3):309–316, 1958.

[Ful90] M. Fulk. Prudence and other conditions on formal language learning. *Information and Computation*, 85:1–11, 1990.

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

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[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.

[JS96] S. Jain and A. Sharma. The intrinsic complexity of language identification. *Journal of Computer and System Sciences*, 52:393–402, 1996.

[Kum89] M. Kummer. A note on direct sums of Friedbergnumberings. *Journal of Symbolic Logic*, 54(3):1009–1010, September 1989.

[Mar89] Y. Marcoux. Composition is almost as good as s-1-1. In *Proceedings, Structure in Complexity Theory–Fourth Annual Conference.* IEEE Computer Society Press, 1989.

[Odi89] P. Odifreddi. *Classical Recursion Theory.* North-Holland, Amsterdam, 1989.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[RC94] J. Royer and J. Case. *Subrecursive programming systems: Complexity & succinctness.* Birkhäuser, 1994.

[Ric80] G. Riccardi. *The Independence of Control Structures in Abstract Programming Systems.* PhD thesis, SUNY/Buffalo, 1980.

[Ric81] G. Riccardi. The independence of control structures in abstract programming systems. *Journal of Computer and System Sciences*, 22:107–143, 1981.

[Rog58] H. Rogers. Gödel numberings of partial recursive functions. *Journal of Symbolic Logic*, 23:331–341, 1958.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, 1967. Reprinted by MIT Press in 1987.

[Roy87] J. Royer. *A Connotational Theory of Program Structure*, volume 273 of *Lecture Notes in Computer Science.* Springer-Verlag, 1987.

[Sto77] J. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[Wei87] K. Weihrauch. *Computability.* Springer-Verlag, 1987.

[Wie78] R. Wiehagen. Characterization problems in the theory of inductive inference. In *Proceedings of the 5th International Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 494–508. Springer-Verlag, 1978.

[Wie96] R. Wiehagen. Characterizations of learnability in various hypothesis spaces. Private communication, 1996.

[ZL95] T. Zeugmann and S. Lange. A guided tour across the boundaries of learning recursive languages. In K. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 190–258. Springer-Verlag, 1995.