Confident Iterative Learning in Computational Learning Theory

Vanja Doskoč*

Vienna University of Technology, Institute of Discrete Mathematics and Geometry,
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria,
vanjadosk@gmail.com

Abstract. In inductive inference various types of learning have emerged. The main aim of this paper is to investigate a new type of learning, the confident iterative learning. Given a class to be learnt, the idea here is to merge the following two concepts. For confidence, we require the learner to converge on any set, however, it only needs to be correct on the sets in the class. To be iterative, we restrict the learner's memory on previous inputs and calculations to its last hypothesis. Investigating the new learner, we will provide negative and positive examples, as well as some properties the confident iterative learner possesses. This will peak at a classification theorem for certain types of classes.
Next, we will introduce and compare different types of confidence, focusing on the learner's behaviour on sets outside of the class. Lastly, we will focus on the possible hypotheses. Introducing learning with respect to hypothesis spaces, we will provide examples witnessing that exact, class preserving and class comprising learning are different.

Keywords: Inductive inference; Confident iterative learning; Very and strongly confident iterative learning; Hypothesis space.

1 Introduction

Given a numbering of all partial computable functions $\varphi_0, \varphi_1, \ldots$, inferring a computable function is the process of finding a code e such that $f = \varphi_e$, when successively fed information on f. The same can be done when given a numbering of all computably enumerable (c.e.) sets $W_0, W_1 \ldots$, where $W_n = \mathrm{dom}(\varphi_n)$. We will infer languages, i.e. **non-empty c.e. sets**, using learning machines, i.e. **computable functions**. One natural form of learning a language is receiving the positive information of the language, i.e. words which are in the language. As suggested in [12], we can compare that to an archaeologist coming across some ruins. There, the archaeologist will find examples of words of the language, rather than counterexamples. This kind of learning is called *learning from text*, see [12]. When inferring a language, one faces several problems. Firstly, no human or computer has infinite memory capabilities. One learner with such a restriction is the iterative learner, see [6] or [8]. Secondly, a learner should be able

* Supported by the Austrian Science Fund FWF through the project P 27527.

to deal with irregular information, i.e. information that does not belong to the language, and still converge on it. A formalization of such a learner is the confident learner, see [3] or [4]. Both learning styles, also in combination with other criteria, have been studied intensively, see for example [3] and [6]. However, the combination of these seems unexplored, yet. We would like to start filling this gap by presenting the results of the respective Master's thesis. This combination may provide additional insight especially for the hard to understand iterative learning, see [6].
In **Section 2**, we establish some examples and general results which hold for confident iterative classes. The most prominent result here is the Classification Theorem 18. To further understand the topic, in **Section 3**, we investigate iterative learning combined with stricter versions of confident learning, gaining the respective hierarchies found in Corollary 25 and Theorem 33. Lastly, in **Section 4**, we consider confident iterative learning with respect to hypothesis spaces, obtaining the hierarchy presented in Theorem 43.
Besides standard notation used in computability theory, see [2] or [11], we will use the notations introduced hereinafter.

Remark 1. For ease of notion, we informally describe programs e, which then could be easily translated into a W-index. We call a function used for this *translation*, see Appendix.

Notation 2. We will write $\forall_y^\infty x : A(x)$, or $\forall^\infty x : A(x)$, if $\exists y\ \forall x \geq y : A(x)$.

Notation 3. Let $\mathcal{S}$ be a class and $D \subseteq \mathbb{N}$. We write $(d_n)_n \in D$ if $d_0, d_1, \dots$ is some *input sequence of* D, i.e. if $\{d_0, d_1, \dots\} = D$.
Furthermore, given some input sequence $d_0, d_1, \dots$, we write $(d_n)_n \in D_{\mathcal{S}}$ if the input belongs to some set $D_{\mathcal{S}}$ in $\mathcal{S}$, i.e. if $\exists D_{\mathcal{S}} \in \mathcal{S} : (d_n)_n \in D_{\mathcal{S}}$. We will call such an input *regular*. Otherwise, or if we allow both cases, we write $(d_n)_n$.

Remark 4. As we will not infer empty sets, we will omit using a pause symbol #.

At every step, the iterative learner has the current datum and its last hypothesis, where information on previous calculations can be stored, at disposal. To formalize that, we will adapt the definition from [6] and [8] to fit our needs.

Definition 5 (IT). A class $\mathcal{S}$ is *iteratively learnable* via $\mathtt{it}(.,.)$ iff, for some fixed initial hypothesis p_0,

$$\forall D \in \mathcal{S}\ \forall (d_n)_n \in D\ \exists p\ \forall^\infty i : \mathtt{it}(p_i, d_i) = p_{i+1} = p \wedge W_p = D.$$

Notation 6. We write $\mathcal{C} \in \mathrm{LC}$ if $\mathcal{C}$ can be learnt via the learning criterion LC.

For confident learning, as explained in [3] or [4], we will allow the learner to be fed any arbitrary input and also request it to converge on it. However, it only has to be correct on regular input.

Definition 7 (CFD). A class $\mathcal{S}$ is *confidently learnable* via $\mathtt{cfd}(.)$ iff

$$\forall (d_n)_n\ \exists e : \Big(\big(\forall^\infty i : \mathtt{cfd}(d_0, d_1, \dots, d_i) = e\big) \wedge \big((d_n)_n \in D_{\mathcal{S}} \Rightarrow W_e = D_{\mathcal{S}}\big)\Big).$$

Definition 8 (CI). A class $\mathcal{S}$ is *confidently iteratively learnable* via $\texttt{ci}(.,.)$ iff $\texttt{ci}(.,.)$ learns $\mathcal{S}$ confidently[1] and iteratively.

2 Confident Iterative Learner

In order to get a feeling for CI-classes consider the following lemma.

Lemma 9. *If $\mathcal{S} \in$ CI, then $\mathcal{S}$ cannot contain an infinite ascending chain.*

Proof. Let $\mathcal{S}$ have a CI-learner $\texttt{ci}(.,.)$. Assume, that $\mathcal{S}$ contains an infinite ascending chain $A_0 \subsetneq A_1 \subsetneq \cdots$. Then

A_0: for some input $d_{0,0}, d_{0,1}, \ldots$ of A_0, $\texttt{ci}(.,.)$ will eventually converge, so

$$\exists p_0 \ \forall^{\infty}_{I_0} j : \texttt{ci}(p_{0,j}, d_{0,j}) = p_{0,j+1} = p_0.$$

A_1: Starting with the previous input $d_{0,0}, d_{0,1}, \ldots, d_{0,I_0}$, and then continuing with some $d_{1,I_0+1}, d_{1,I_0+2}, \ldots$ of $A_1 \setminus \{d_{0,0}, d_{0,1}, \ldots, d_{0,I_0}\}$, $\texttt{ci}(.,.)$ will eventually converge, so

$$\exists p_1 \ \forall^{\infty}_{I_1} j : \texttt{ci}(p_{*,j}, d_{*,j}) = p_{*,j+1} = p_1.$$

A_i: Starting with the input $d_{0,0}, \ldots, d_{0,I_0}, d_{1,I_0+1}, \ldots, d_{1,I_1}, d_{2,I_1+1}, \ldots, d_{i-1,I_{i-1}}$, and continuing with some $d_{i,I_{i-1}+1}, d_{i,I_{i-1}+2}, \ldots$ of $A_i \setminus \{d_{0,I_0}, \ldots, d_{i-1,I_{i-1}}\}$, $\texttt{ci}(.,.)$ will eventually converge, so

$$\exists p_i \ \forall^{\infty}_{I_i} j : \texttt{ci}(p_{*,j}, d_{*,j}) = p_{*,j+1} = p_i.$$

So, we have an input sequence

$$d_{0,0}, \ldots, d_{0,I_0}, d_{1,I_0+1}, \ldots, d_{1,I_1}, d_{2,I_1+1}, \ldots, d_{i-1,I_{i-1}}, d_{i,I_{i-1}+1}, \ldots$$

where $\texttt{ci}(.,.)$ outputs the codes $\ldots, p_0, \ldots, p_1, \ldots, p_i, \ldots$, which are all different. So, $\texttt{ci}(.,.)$ does not converge on this input of $A := \bigcup_i A_i$, a contradiction. □

Now, we will show that the other direction does not hold.

Example 10. Let $\mathbb{N}_e$ be the set of all even natural numbers, and let $\mathbb{N}_o$ be the set of all odd natural numbers. Consider the class

$$\mathcal{C} := \{E \mid \exists n : 2n+1 \in E \wedge |\mathbb{N}_e \cap E| = 2n \wedge |\mathbb{N}_o \cap E| = 1\}.$$

The sets of this class contain one odd element $2n+1$ and $2n$ many even elements. One can show that $\mathcal{C} \in$ CFD and $\mathcal{C} \in$ IT, see Appendix.
Now, we will prove that $\mathcal{C}$ is not CI. Assume there exists a CI-learner $\texttt{ci}(.,.)$. Consider some input sequence $(e_n)_n \in \mathbb{N}_e$. The learner $\texttt{ci}(.,.)$ has to converge on that input, so

$$\exists p \ \forall^{\infty}_{I} j : \texttt{ci}(p_j, e_j) = p_{j+1} = p.$$

[1] Confidence is only required for the argument of the input sequence.

Now consider the set $L_I = \{e_0, e_1, \ldots, e_I\}$. Without loss of generality, we can assume that $|L_I| = 2k$ for some $k \in \omega$.
If we take four different even numbers that have not appeared in the input, yet, i.e. $n_1, \ldots, n_4 \in \mathbb{N}_e \setminus L_I$, then $N_1 := L_I \cup \{n_1, n_2\}$ and $N_2 := L_I \cup \{n_3, n_4\}$ still have the same code p, as $\texttt{ci}(.,.)$ will not change its mind on even numbers anymore. Furthermore, $|N_1| = |N_2| = 2(k+1)$. So, if we consider $N_1 \cup \{m\}$ and $N_2 \cup \{m\}$, where $m = 2(k+1)+1$, the sets appear to be in $\mathcal{C}$. However, these two sets have the same code $\texttt{ci}(p, m) = p'$. A contradiction.

So we see that having an iterative and a confident learner does not suffice to have a confident iterative learner. To see another example, we need the following lemma.

Lemma 11. *Let $\mathcal{C} \in$ CI. Then, for every finite set $F \in \mathcal{C}$ we can compute its code, i.e. for certain input $(d_n)_n \in F$ we can find I effectively such that for some p we have $\texttt{ci}(p_i, d_i) = p_{i+1} = p$ for all $i > I$.*

Proof. Let $\mathcal{C} \in$ CI via $\texttt{ci}(.,.)$, and $F = \{f_0, \ldots, f_n\} \in \mathcal{C}$. Now, we will feed $\texttt{ci}(.,.)$ the input $f_0, f_1, \ldots, f_n, f_n, f_n, \ldots$, abbreviated by $(f_m^*)_m$.
As the class is CI, $\texttt{ci}(.,.)$ will learn the set on this input, i.e. there exists some p and I such that $\texttt{ci}(p_j, f_j^*) = p_{j+1} = p$ for all $j > I$. Without loss of generality, let $I > n$. Since the elements of the input sequence do not change anymore, once $\texttt{ci}(.,.)$ repeats its output, it actually learned the set, since the next calculation is the same as the previous. I.e. at some point $I > n$ the computation outputs $\texttt{ci}(p_I, f_I^*) = \texttt{ci}(p_I, f_n) = p_I$, thus receives the same input again, namely p_I and f_n. Thus, $\forall j > I : \texttt{ci}(p_j, f_j^*) = \texttt{ci}(p_j, f_n) = p_j$. That I is the sought index. □

Example 12. Consider the class $\mathcal{M} = \{A \mid \exists x, y : A \subseteq M_{x,y}\}$ where, with K being the halting set,

$$M_{x,y} := \{2^x, 2^x 3, \ldots, 2^x 3^y : x \text{ is enumerated into } K \text{ in exactly } y \text{ steps}\}.$$

Again, one can show that $\mathcal{M} \in$ CFD and $\mathcal{M} \in$ IT, see Appendix.
Now, assume $\mathcal{M}$ has a CI-learner $\texttt{ci}(.,.)$. For $x \in \mathbb{N}$, consider the following computation. Let $(x \in K)_y$ be the abbreviation for the computable question, whether x is enumerated into K in at most y steps.

0: Input $2^x, \ldots, 2^x$ until $\texttt{ci}(.,.)$ computably converges, see Lemma 11, to some p_1. Then, check if $(x \in K)_0$. If so, then $x \in K$. Otherwise, proceed with the next step.
1: Continue to input $2^x 3, \ldots, 2^x 3$ until $\texttt{ci}(.,.)$ computably converges to some p_2. Then, check if $p_1 = p_2 \vee (x \in K)_1$. In the second case, again $x \in K$. In the first case, $W_{p_1} = W_{p_2}$. So, if $x \in K$, then the class members $\{2^x\}$ and $\{2^x, 2^x 3\}$ would have the same code. A contradiction. Thus, $x \notin K$. Otherwise, proceed with the next step.
i: Continue to input $2^x 3^i, \ldots, 2^x 3^i$ until $\texttt{ci}(.,.)$ computably converges to some p_{i+1}. Then, check if $p_i = p_{i+1} \vee (x \in K)_i$. If not, then again $x \notin K$ or $x \in K$, respectively. Otherwise, proceed with the next step.

This computation has to stop, as otherwise, as $p_i = p_{i-1}$ is one of the requirements, $\texttt{ci}(.,.)$ would not converge on the input sequence

$$2^x, \ldots, 2^x, 2^x 3, \ldots, 2^x 3, 2^x 3^2, \ldots, 2^x 3^{i-1}, 2^x 3^i, \ldots.$$

Thus, the computation stops for all x, and therefore we can solve the halting problem, a contradiction.

So far, we have only seen classes that are not CI. To get rid of this peculiar situation, we will provide some examples of CI classes.

Example 13. For $n \in \mathbb{N}_{>0}$ let $\mathcal{C}_n := \{A : |A| \leq n\}$. Then, $\mathcal{C}_n \in$ CI, see Appendix.

This example is of some finite nature, as all the sets do not expand some bound. However, we can find examples with a set of every size. The following remark and lemma will help us to do so.

Remark 14. In our algorithms, we will use "output x", meaning that the algorithm outputs x as its current hypothesis and then requests the next input, and "return 0", implying that the computation could actually halt here.

Lemma 15. *Let $R(x, y)$ be a two place computable relation, such that*

$$\forall x : S_{R(x)} := \{2^x 3^y : R(x, y), y \in \mathbb{N}\} \text{ is finite.} \tag{1}$$

Then the class $\mathcal{C}_R := \{S \mid \exists x : S \subseteq S_{R(x)}\}$ is CI.

Proof. Let $R(x, y)$ be a two place computable relation such that (1). Then, for some input sequence $(d_n)_n$ the following algorithm will learn $\mathcal{C}_R$ confidently iteratively. Let $\exp(x, y)$ be the computable function that outputs the exponent of the x-th prime number in the prime decomposition of y. Let p_0 be a program of the function f, where $f(0) = \exp(0, d_0)$ and $f(x + 1) = 0$ for $x \geq 0$. Also, let $\overline{\text{sgn}}(x) := 1 - \text{sgn}(x)$. Then, execute the following algorithm.

1. For input d_i and p_i, check whether $d_i = 2^{s_i} 3^{t_i}$. If so, then
2. for $s = \varphi_{p_i}(0)$, check whether $s = s_i$. If so, then
3. check whether $\varphi_{p_i}(d_i + 1) = 0$. If so, then
4. check whether $R(s_i, t_i)$. If so, then
5. let p_{i+1} be a program of $f(x) = \varphi_{p_i}(x) + \overline{\text{sgn}}\,(|x - (d_i + 1)|)$.
6. In the else case of step 3 output $p_{i+1} = p_i$. In all remaining else cases, return 0.

Notice, that p_{i+1} in step 5 can be computed using p_i, see [11]. For some input p_i and d_i, we first check the form, step 1, and whether $s = s_i$, step 2. Then we check whether the datum is new, step 3, and whether the condition $R(s_i, t_i)$ is met, step 4. If all of this is true, output a code of the corresponding characteristic function f. Else, one can return 0, as the input cannot belong to some set in the class. The only exception here is when the element is already seen, i.e. step 3, where the algorithm outputs the last hypothesis.
Since every $S_{R(x)}$ is finite, the algorithm may only come to step 5 finitely many times. If the input is not regular, at some time the algorithm will return 0. Thus, it works properly. One can recompute $S_{R(x)}$ via $\chi_{S_{R(x)}}(x) = \varphi_p(x + 1)$. □

Example 16. By Lemma 15, $\mathcal{C} := \{S \mid \exists x : S \subseteq S_x\} \in \mathrm{CI}$, where

$$S_x := \{2^x, 2^x 3, \ldots, 2^x 3^y : y \leq x\}.$$

So far, we have gathered examples of CI classes and such that are not. Next, we aim to find a classification of some confident iterative classes.

Notation 17. Let $D_0, D_1, \ldots$ be an effective enumeration of all finite sets. Then, for $x \in \mathbb{N}$ we call D_x the *x-th finite set*.

Theorem 18 (Classification Theorem). *Let $\mathcal{C}$ be a class such that*

1. *$\mathcal{C}$ is a class of finite sets,*
2. *$\mathcal{C}$ is closed under subsets,*
3. *there exists an effective procedure which tells whether $D_x \in \mathcal{C}$.*

Then, $\mathcal{C}$ is CI *iff $\mathcal{C}$ does not contain an infinite ascending chain.*

Proof. Let $\mathcal{C}$ be as stated.

$\Rightarrow$: By Lemma 9, if $\mathcal{C} \in \mathrm{CI}$, then $\mathcal{C}$ cannot contain an infinite ascending chain.

$\Leftarrow$: Let $\mathcal{C}$ have no infinite ascending chain. Also, let $(d_n)_n$ be any input sequence. Then, the following algorithm will serve as a CI-learner. Let p_0 be a program of the zero function.

1. For some input d_i and p_i, if $\varphi_{p_i}(d_i + 1) = 0$, then
   (a) collect all $\varphi_{p_i}(0) = n$ many elements that have appeared so far, i.e. $O_i = \{x : \varphi_{p_i}(x + 1) = 1\}$ with $|O_i| = n$. Then
   (b) check whether $O_i \cup \{d_i\} = D_{x'} \in \mathcal{C}$.
      i. If so, let p_{i+1} be a program of

$$f(x) = \varphi_{p_i}(x)\,\mathrm{sgn}(x) + (\varphi_{p_i}(x) + 1)\,\overline{\mathrm{sgn}}(x) + \overline{\mathrm{sgn}}\,(|x - (d_i + 1)|)\,.$$

      ii. Else, return $p_{i+1} = 0$.
2. Else, $p_{i+1} = p_i$.

The only step that is not obviously computable, is finding O_i. Since the algorithm knows how many elements O_i has to contain, one can conduct the search effectively. Thus, all steps in the algorithm can be done effectively.
For some input $(d_n)_n$, the algorithm checks whether d_i is some new input, i.e. $\varphi_{p_i}(d_i + 1) = 0$. If so, it collects all the information so far, and then checks whether the corresponding set $D_{x'}$ belongs to the class. If it does, it outputs a program of its characteristic function, with an additional update on the first argument. Else, i.e. if the algorithm witnesses $D_{x'}$ not belonging to the class, it returns 0.
So, the only case where the machine could possibly change its mind infinitely many times, is when it confirms $D_x \in \mathcal{C}$ and thus changes the program. However, as $\mathcal{C}$ has no infinite ascending chain, this cannot happen. □

3 Advanced Confidence

In this section we will expand the idea of the confident learner. As discussed, the confident learner converges even if the input does not belong to any set in the class. However, we can request it to output some fixed dummy code, i.e. -1, if the learner detects such a case. With that idea in mind, we introduce the very confident learner. A similar attempt can be found in [7].

Definition 19 (vC). A class $\mathcal{C}$ is *very confidently learnable* via $\mathtt{vc}(.)$ iff

$$\forall (d_n)_n : \exists e \Big(\big(\forall^\infty i : \mathtt{vc}(d_0, \ldots, d_i) = e\big) \wedge \big((d_n)_n \in D_\mathcal{C} \Rightarrow W_e = D_\mathcal{C}\big) \Big) \wedge$$
$$\forall j \Big(\big(\forall S \in \mathcal{C} : \{d_0, \ldots, d_j\} \neq S\big) \Rightarrow \big(\mathtt{vc}(d_0, \ldots, d_j) = -1\big) \Big).$$

Remark 20. Classes in vC cannot have any infinite sets, see Appendix.

By requesting the learner to stop its computations once it detects some irregular input, we can get that idea even further.

Definition 21 (sC). A class $\mathcal{C}$ is *strongly confidently learnable* via $\mathtt{sc}(.)$ iff

$$\forall (d_n)_n : \exists e \Big(\big(\forall^\infty i : \mathtt{sc}(d_0, \ldots, d_i) = e\big) \wedge \big((d_n)_n \in D_\mathcal{C} \Rightarrow W_e = D_\mathcal{C}\big) \Big) \wedge$$
$$\forall J \Big(\big(\forall S \in \mathcal{C} : \{d_0, \ldots, d_J\} \neq S\big) \Rightarrow \big(\forall j \geq J : \mathtt{sc}(d_0, \ldots, d_j) = -1\big) \Big).$$

Corollary 22. sC $\subseteq$ vC $\subseteq$ CFD.

Corollary 22 follows directly from the definitions. The following examples show that the inclusions are proper.

Example 23. With K'' being the second jump of the halting set K, let

$$\mathcal{C} = \{\{x\} : x \in \mathbb{N} \wedge x \in K''\}.$$

Then, $\mathcal{C}$ is obviously in CFD, via the confident learner $\mathtt{cfd}(d_0, \ldots, d_i)$ which outputs a code of $\{d_0\}$.
Assume $\mathcal{C} \in$ vC via $\mathtt{vc}(.)$. For $x \in \mathbb{N}$ and the input sequence $x, x, x, \ldots$ we have

$$x \notin K'' \Rightarrow \forall N : \mathtt{vc}(x^N) = -1 \Rightarrow \forall N\ \exists n : (n > N \wedge \mathtt{vc}(x^n) = -1),$$
$$x \in K'' \Rightarrow \exists N\ \forall n : (n > N \Rightarrow \mathtt{vc}(x^n) \neq -1).$$

Thus, $x \in K'' \Leftrightarrow \exists N\ \forall n : (n > N \Rightarrow \mathtt{vc}(x^n) \neq -1)$ and therefore $K'' \leq_m K'$. A contradiction.

Example 24. For $x, y \in \mathbb{N}, x \neq y$, let $\mathcal{C} = \{\{x, y\}\}$. Obviously, $\mathcal{C} \in$ vC. However, it is not in sC. Assume the opposite, with a learner $\mathtt{sc}(.)$. Then for some input sequence $x, y, y, \ldots$ the machine $\mathtt{sc}(.)$ will see that $\{x\}$ is not in the class, thus output -1 for everything that is to come, i.e. $\forall j : \mathtt{sc}(x, y^j) = -1$. This is obviously the wrong behaviour, since $\{x, y\}$ is in the class.

Corollary 25. sC $\subsetneq$ vC $\subsetneq$ CFD.

The strongly confident learner may seem awfully weak compared to the very confident one. However, its weakness in the last example originates from the fact, that the class was not closed under subsets. When combining those attempts with the iterative learner, we will see a slightly different behaviour.

Definition 26 (vCI). A class $\mathcal{S}$ is *very confidently iteratively learnable* via $\texttt{vci}(.,.)$ iff $\texttt{vci}(.,.)$ learns $\mathcal{S}$ very confidently[2] and iteratively.

Definition 27 (sCI). A class $\mathcal{S}$ is *strongly confidently iteratively learnable* via $\texttt{sci}(.,.)$ iff $\texttt{sci}(.,.)$ learns $\mathcal{S}$ strongly confidently[2] and iteratively.

Corollary 28. sCI $\subseteq$ vCI $\subseteq$ CI.

Again, Corollary 28 follows directly from the definitions. However, these classes do behave different than the ones in Corollary 25, as we will show next. To do so, we need some auxiliary results.

Lemma 29. *Let $\mathcal{C} \in$ vCI via $\texttt{vci}(.,.)$, and let $(d_n)_n$ be some input.*
Then, if $\texttt{vci}(p_i, d_i) = -1$ for some $i \in \mathbb{N}$, then $\texttt{vci}(p_j, d_j) = -1$ for all $j \geq i$.

Proof. Let $\mathcal{C} \in$ vCI via $\texttt{vci}(.,.)$, let $(d_n)_n$ be some input and let i be such that $\texttt{vci}(p_i, d_i) = -1$. Assume that $\exists j > i : \texttt{vci}(p_j, d_j) \neq -1$. Then,

$$S_1 = \{d_0, \ldots, d_i\} \subsetneq \{d_0, \ldots, d_j\} = S_2$$

as $S_1 \notin \mathcal{C}$, while $S_2 \in \mathcal{C}$. As $\mathcal{C} \in$ vCI $\subseteq$ CI, it cannot contain any infinite ascending chain. So, there is some finite $S_3 \supsetneq S_2$ such that $S_3 \notin \mathcal{C}$. Now, consider the following two input sequences

$$d_0, \ldots, d_i, d_{i+1}, \ldots, d_j, d^*_{j+1}, \ldots,$$
$$e_0, \ldots, e_m, d_{i+1}, \ldots, d_j, d^*_{j+1}, \ldots,$$

where $d^*_k \in S_2$, and $e_k \in S_3$, such that $\{e_0, \ldots, e_m\} = S_3$ and $\texttt{vci}(p_m, e_m) = -1$. Then, as the input is the same after the occurrence of -1, $\texttt{vci}(.,.)$ converges to the same program on both inputs. This is a contradiction, since the first input belongs to $S_2 \in \mathcal{C}$ and the second to $S_3 \notin \mathcal{C}$. □

In particular, the previous lemma also shows the next results.

Corollary 30. *Every class $\mathcal{C} \in$ vCI is closed under subsets.*

Lemma 31. *Let $\mathcal{C}$ be a class. Then, $\mathcal{C} \in$ vCI if and only if $\mathcal{C} \in$ sCI.*

[2] An analogous version of Footnote 1 applies here.

Proof. The right to left direction follows from Corollary 28.
For the other direction, let $\mathcal{C} \in$ vCI. Then, for some input sequence $(d_n)_n$ and the same starting program p_0, let $\texttt{sci}(p_i, d_i) = \texttt{vci}(p_i, d_i)$.
It only remains to prove that for any J

$$\big(\forall S \in \mathcal{C} : \{d_0, \ldots, d_J\} \neq S\big) \Rightarrow \big(\forall j \geq J : \texttt{sci}(p_j, d_j) = p_{j+1} = -1\big).$$

Via Lemma 29, for any $J \in \mathbb{N}$,

$$\big(\forall S \in \mathcal{C} : \{d_0, \ldots, d_J\} \neq S\big) \overset{\texttt{vci}(p_J, d_J) = -1}{\Rightarrow} \big(\forall j \geq J : \texttt{vci}(p_j, d_j) = -1\big).$$

Since $\texttt{vci}(.,.) = \texttt{sci}(.,.)$, we have the sought learner. □

This lemma is not too surprising. Since the only information on the previous calculations and inputs has to be coded into the output in some form, outputting -1 deletes all that information. Thus, it is the same as stopping.

Theorem 32. *Let $\mathcal{C}$ be a class of finite sets. Then $\mathcal{C} \in$ vCI if and only if $\mathcal{C} \in$ CI, $\mathcal{C}$ is closed under subsets, and there is a decision procedure telling whether or not $D_x \in \mathcal{C}$.*

Proof. We will prove each direction separately.

⇒: If $\mathcal{C} \in$ vCI, then, by Corollary 28, $\mathcal{C} \in$ CI, and by Corollary 30 the class has to be closed under subsets. For $x \in \mathbb{N}$, we can compute a code p of D_x effectively, see Lemma 11. Then, $D_x \in \mathcal{C} \Leftrightarrow p \neq -1$.

⇐: For the other direction, let $\mathcal{C} \in$ CI via $\texttt{ci}(.,.)$ and the starting program $\tilde{p_0}$, as well as some translation $s(.)$, see Remark 1. Let $\mathcal{C}$ be closed under subsets and have a decision procedure for $D_x \in \mathcal{C}$. Let $c(.)$ be some coding of finite sets. Then, for the starting program $p_0 = 2^{c(\emptyset)}3^{\tilde{p_0}}$,

$$\texttt{vci}(p_i, d_i) = \begin{cases} 2^{c(\{d_0,\ldots,d_i\})}3^{\texttt{ci}(p_i,d_i)}, & \text{if } \{d_0, \ldots, d_i\} \in \mathcal{C} \wedge p_i \neq -1, \\ -1, & \text{else.} \end{cases}$$

will do the trick. Here $s(\exp(1,.))$ is the translation.
Let $(d_n)_n$ be some input sequence. For input d_i and p_i, the machine checks whether or not $c^{-1}(\exp(0, p_i)) \cup \{d_i\} = \{d_0, \ldots, d_{i-1}, d_i\} \in \mathcal{C}$. If not, it outputs -1, and after that never changes its mind again. Otherwise, it will compute the code that $\texttt{ci}(p_i, d_i)$ would have, and output $2^{c(\{d_0,\ldots,d_i\})}3^{\texttt{ci}(p_i,d_i)}$.
As $\texttt{ci}(.,.)$ learns this class, and as it cannot contain an infinite ascending chain, see for the Classification Theorem 18, the machine may only change its mind finitely often.
Again, as $\texttt{ci}(.,.)$ learns the class correctly, it will converge on any $(d_n)_n$ to some $\tilde{p}$. If $(d_n)_n \in S_{\mathcal{C}}$, then the program $\tilde{p}$ has to be correct, i.e. $W_{s(\tilde{p})} = S_{\mathcal{C}}$.
If we compute $s(\exp(1, p)) = s(\tilde{p})$, we get the computable $s(\exp(1,.))$ as translation. So, the algorithm will always converge and behave correctly. □

As classes in CI do not need to be closed under subsets, we get the following.

Theorem 33. sCI = vCI $\subsetneq$ CI.

4 Learning with Hypothesis Spaces

Until now, we did not care too much about the hypotheses, as long as they gave some sort of computable information on the learnt set. However, it did prove to be inconvenient to code all the information into the hypotheses itself. We also witnessed that for certain problems, certain hypotheses were more comfortable to use. To drill down onto that, we will equip the classes with hypothesis spaces. To do so, we need to introduce the following notion, see for example [1] or [5].

Definition 34. A class $\mathcal{C}$ is given by a *uniformly indexed family* if there exists a two-place, $\{0,1\}$-valued, computable function L, such that

1. $L(e,x) = L_e(x) = \begin{cases} 1, & x \in L_e, \\ 0, & x \notin L_e. \end{cases}$
2. $\forall e : L_e \in \mathcal{C}$,
3. $\forall L \in \mathcal{C}\ \exists e : L = L_e$.

We will call such a class $\mathcal{C}$ *indexed class* for short. We will denote these by $\mathcal{C} = \{L_e : e \in \mathbb{N}\}$, where $L_e = \{x : L_e(x) = 1\}$.

Remark 35. Since we are only considering classes without the empty set, we additionally demand that none of the L_e is empty.

So, indexed classes provide an effective enumeration of the class and also an effective way to check whether $x \in L_e$ for some x, e. With this, we can introduce the idea of the hypothesis space.
Given some indexed class $\mathcal{C}$, we will learn it with respect to some *hypothesis space* $\mathcal{H} = \{H_i : i \in \mathbb{N}\}$, which itself is an indexed class. Learning a set $C \in \mathcal{C}$ here means that the learning machine converges to some i, such that $C = H_i$. In [5] or [9], we can find three different ideas, how such learning can be done.

- Exactly: $\mathcal{C}$ is *exactly learnable*, if $\mathcal{H} = \mathcal{C}$, using the same numbering.
- Class Preservingly: $\mathcal{C}$ is *class preservingly learnable*, if $\mathcal{H} = \mathcal{C}$.
- Class Comprisingly: $\mathcal{C}$ is *class comprisingly learnable*, if $\mathcal{H} \supseteq \mathcal{C}$.

Notation 36. Contrary to the widely used E, ϵ and C, we will use the prefixes E, CP and CC, respectively. Also, we will write $\mathcal{C} \in \text{LC}(\mathcal{H})$ if the class $\mathcal{C}$ can be learnt via the learning criterion LC with respect to the hypothesis space $\mathcal{H}$.

Remark 37. Since the hypotheses are restricted now, we will use an "initial state" as first hypothesis p_0 to signalize the first computation step, rather than a proper hypothesis, see [6]. However, the learner may never output this state.

Corollary 38. $\text{ECI} \subseteq \text{CPCI} \subseteq \text{CCCI}$.

Corollary 38 follows directly from the definitions. In order to investigate learning with respect to some hypothesis space and to show that the inclusions are proper, let us fix one special indexation of all non-empty, finite sets.

Definition 39. Let $[i]_2$ be the binary expression of i and $\mathcal{B} = \{B_i : i > 0\}$, with

$$B_i(x) = ([i]_2)(x) = \begin{cases} 1, & \text{if } [i]_2 \text{ is 1 at position } x, \\ 0, & \text{else.} \end{cases}$$

Lemma 40. *Let $\mathcal{C} = \{C_i : i \in \mathbb{N}\}$ be an indexed class of finite sets, which is closed under subsets, with a decision procedure for $D_x \in \mathcal{C}$. Then*

$$\mathcal{C} \in \text{CCCI} \Leftrightarrow \mathcal{C} \text{ contains no infinite ascending chain} \Leftrightarrow \mathcal{C} \in \text{CI}.$$

Proof. Since the second equivalence is exactly the Classification Theorem 18, we only need to show the first one.

$\Rightarrow$: Conducting some similar proof as in the proof of Lemma 9, we can see that this direction is true.

$\Leftarrow$: By observing the proof of the Classification Theorem 18, we can see that we could change the output to its respective counterpart in $\mathcal{B}$, and still receive a natural bound on the amount of the elements. Thus, we could conduct the same proof here. □

Example 41. Consider the class $\mathcal{C} = \{C_{x,y} : x, y \in \mathbb{N}\}$, where

$$C_{x,y} = \begin{cases} \{p\}, & x \notin K \text{ in } y \text{ steps}, \\ \{x\}, & x \in K \text{ in } y \text{ steps}. \end{cases}$$

where p is a program of some everywhere undefined function.
By outputting 2^{d_0} on the input $(d_n)_n$, we see that $\mathcal{C} \in \text{CCCI}(\mathcal{B})$.
Assume, $\mathcal{C} \in \text{CPCI}(\mathcal{H})$. For $x \in \mathbb{N}$ consider the input sequence $x, x, \ldots$. At some computable stage N, the learner `cpci`$(.,.)$ will converge to some p_x, see Lemma 11. Then, for $x \neq p$,

$$x \in K \Rightarrow H_{p_x} = \{x\},$$
$$x \notin K \Rightarrow H_{p_x} \neq \{x\}.$$

The second property originates from the fact, that there is no set in $\mathcal{H}$ representing $\{x\}$, since $x \notin K$. For any i and x, the question whether $H_i = \{x\}$ is computable, as all H_i have exactly one element. Thus, the canonical search for y such that $H_i(y) = 1$ will terminate.
Thus, $x \in K \Leftrightarrow H_{p_x} = \{x\}$, with the latter being computable. A contradiction.

The next example is inspired by an example in [5] and [10].

Example 42. Consider the following indexed classes $\mathcal{L} = \{L_{i,j} : i, j \in \mathbb{N}\}$ and $\tilde{\mathcal{L}} = \{\tilde{L}_{i,j} : i, j \in \mathbb{N}\}$, where

$$\tilde{L}_{i,j} = \begin{cases} \{2^i 3^n : n \in \mathbb{N}\}, & (\varphi_i(i))_j \uparrow, \\ \{2^i 3^n : n \leq k\}, & (\varphi_i(i))_j \downarrow \text{ in } k \text{ steps}. \end{cases}$$

$$L_{i,j} = \begin{cases} \tilde{L}_{i,0}, & j = 0, \\ \tilde{L}_{i,j}, & j > 0 \text{ and } (\varphi_i(i))_j \downarrow \text{ in exactly } j \text{ steps}, \\ \tilde{L}_{0,0}, & \text{else.} \end{cases}$$

Notice that $\mathcal{L} = \tilde{\mathcal{L}}$. Without loss of generality, assume that every program needs at least two steps to compute, i.e. $\tilde{L}_{i,0} = \tilde{L}_{i,1}$ are infinite for all $i \in \mathbb{N}$. For input $(d_n)_n$ and for some starting program $p_0 = (s_0, 0)$, where $s_0 = \exp(d_0, 0)$, the following algorithm will show that $\mathcal{L} \in \text{CPCI}(\tilde{\mathcal{L}})$.

1. For d_i and p_i, check whether $d_i = 2^{s_i}3^{t_i}$ and whether $p_i = (s_i, x)$. If not, return 0.
2. If so, check whether $(\varphi_{s_i}(s_i))_{t_i} \downarrow$ and let k be the amount of steps needed.
   (a) If $x = 0$ and $t_i = k$, output (s_i, t_i). If $t_i > k$, output $(s_i, 1)$.
   (b) If $x = 1$, output $p_{i+1} = p_i$.
   (c) If $x > 1$, and if $t_i > k$, output $(s_i, 1)$. Else, output $p_{i+1} = p_i$.
   (d) If not $(\varphi_{s_i}(s_i))_{t_i} \downarrow$, output $p_{i+1} = p_i$.

One can easily see that the algorithm works correctly. Let $(d_n)_n$ be some input, and let p_i and d_i be the current input. Then, after checking the form, we check whether $p_i = (s_i, x)$ and $\varphi_i(i) \downarrow$. If $x = 0$ and the function converged, we output (s_i, t_i) if $t_i = k$, as the input belongs to a finite set, and output $(s_i, 1)$, as the input outranged the finite set. If $x = 1$, we already witnessed that second case, and thus just output the previous hypothesis. If $x > 1$, we already witnessed $\varphi_i(i)$ to converge, so we only change mind if $t_i > k$, i.e. if we outranged a finite set again. If we did not witness $\varphi_i(i)$ to converge, we simply output the previous hypothesis and request the next input.
Assume now, that $\mathcal{L} \in \text{ECI}$ via $\texttt{eci}(.,.)$. Let $i \in \mathbb{N}_{>0}$, and consider some canonical input $(d_n)_n \in L_{i,0}$, i.e. $d_j = 2^i 3^j$. At some point m, $\texttt{eci}(.,.)$ has to output $(i, 0)$. Now check whether $\varphi_i(i) \downarrow$ in at most m steps. If so, then $\varphi_i(i)$ converges. If not, then $\varphi_i(i)$ has to diverge. Assume the opposite, i.e. $\varphi_i(i)$ converges at some later point $m' > m$. Then, on the input $2^i, 2^i 3, \ldots, 2^i 3^{m'}, 2^i 3^{m'}, \ldots$, $\texttt{eci}(.,.)$ will computably converge to (i, m'). If we continue to input $2^i 3^{m'+1}, 2^i 3^{m'+2}, \ldots$ once $\texttt{eci}(.,.)$ converged, at some point we will witness $(i, 0)$ as output again. Now, we can repeat the input from when we witnessed $(i, 0)$ for the first time. Thus, the machine will not converge on this input, a contradiction. So, we can solve the halting problem, a contradiction.

Theorem 43. $\text{ECI} \subsetneq \text{CPCI} \subsetneq \text{CCCI}$.

Acknowledgments. I would like to thank my supervisor Ekaterina Fokina for her excellent support and patience.

Appendix

Addition (for Remark 1). Intuitively, a program of a set S should provide us with the information which elements are in the set. For example, if we know the characteristic function χ_S, we can compute the set. However, that is not consistent with our definition of a program of S. Since $\chi_S = \varphi_e$ is everywhere defined, $W_e = \mathbb{N}$. To circumvent that, we will allow reinterpretations of programs.
Using that and letting $f(p, m, x) = \varphi_p(x + m)$ if latter is 1, and undefined else, we can prove that having the shifted version of the characteristic function χ_S allows us to compute the set. For fixed m, let e be such that $\varphi_e(x+m) = \chi_S(x)$. Then, using the S_n^m Theorem on $f = \varphi_n$, we obtain

$$\varphi_{s(n,p,m)}(x) = \varphi_n(p, m, x) = f(p, m, x) = \begin{cases} 1, & \varphi_p(x + m) = 1, \\ \uparrow, & \text{else.} \end{cases}$$

Then, $W_{s(n,e,m)} = S$.

Addition (for Example 10). First, observe that any subclass $\mathcal{S}$ of $\mathcal{P}_{fin}(\omega) = \{A \subseteq \omega : |A| < \infty\}$ can be learned iteratively.

Proof. Let $\mathcal{S}$ be a subclass of $\mathcal{P}_{fin}(\omega)$ and $(d_n)_n \in A_{\mathcal{S}}$ the input for the algorithm below. Let p_0 be the program of the zero function. For inputs p_i and d_i,

1. if $\varphi_{p_i}(d_i) = 0$, then p_{i+1} is a program of $f(x) = \varphi_{p_i}(x) + \overline{\text{sgn}}(|x - d_i|)$,
2. otherwise, $p_{i+1} = p_i$.

Let $(d_n)_n \in A_{\mathcal{S}}$ be some input sequence. For any input d_i the algorithm checks, whether d_i is a new datum, i.e. $\varphi_{p_i}(d_i) = 0$. In the first case, it changes the characteristic function on this argument to 1, leading to $f(x)$. Then it computes and outputs a code of $f(x)$. Otherwise, it outputs the last hypothesis.
At some point I, all elements of $A_{\mathcal{S}}$ have appeared in the input, i.e. $\forall_I^\infty i : \{d_0, \ldots, d_I\} = \{d_0, \ldots, d_i\} = A_{\mathcal{S}}$. Now, the algorithm will always proceed with step 2, as there are no yet unmentioned elements anymore.
So, the algorithm will converge, and output a code of the characteristic function of $A_{\mathcal{S}}$. □

Thus, $\mathcal{C}$ is iteratively learnable.
Next, we will provide a confident learner of $\mathcal{C}$. For any input $(d_n)_n$, execute the following algorithm.

1. If $\{d_0, \ldots, d_i\}$ contains no odd elements, output 0.
2. If $\{d_0, \ldots, d_i\}$ contains one odd element $m = 2n + 1$ and $2n$ many even elements, output a code of the characteristic function of $\{d_0, \ldots, d_i\}$.
3. Else, output 0.

To see that the algorithm works properly, let $(d_n)_n$ be some input sequence. The idea of the algorithm is simple, we count the odd elements, leading us into three cases on $\{d_0, \ldots, d_i\}$.

1. While we have no odd elements, we do not have a set in the class, yet, so we output any dummy code, i.e. 0.
2. When we have exactly one odd element $m = 2n + 1$ and $2n$ many even elements, we output a code of the characteristic function of $\{d_0, \ldots, d_i\}$, as it is a set in the class.
3. Else, output 0.

Since the algorithm does only change its mind when changing from case 1 or 3 to 2 or from 2 to 3, which can only happen finitely many times, we see that the algorithm works properly.

Addition (for Example 12). As above, $\mathcal{M} \in \mathrm{IT}$.
In order to show that the class is confidently learnable, recall the computable function $\exp(x, y)$, which outputs the exponent of the x-prime number in the prime decomposition of y. Also, let $[x \in K]_y$ output the number of steps $\leq y$ needed for x to be enumerated into K. If x is not enumerated into K in y steps, let that function output $y + 1$.
Now, let $(d_n)_n$ be some input for the algorithm. Upon starting, fix two global variables, namely $b = 0$ and $s = \exp(0, d_0)$.

1. For input d_i and p_i, check whether $d_i = 2^{s_i}3^{t_i}$. If not, return 0. Else,
2. check whether $s = s_i$. If not, return 0. If so,
3. calculate $t = [s_i \in K]_{t_i}$. Now compare t and t_i.

$t < t_i$: Return 0.
$t = t_i$: Output a program of the characteristic function of $\{d_0, \ldots, d_i\}$.
$t > t_i$: For[3] $b = \max(b + 1, t)$, calculate $t' = [s_i \in K]_b$.
  $t' \leq b$: Output a program of the characteristic function of $\{d_0, \ldots, d_i\}$.
  $t' > b$: Output 0, but request the next input.

We will prove that the algorithm works as supposed and with that explain it. Let $(d_n)_n$ be some input. As already mentioned, we have two global variables. First, we have a counter b which will come to use later. Secondly, we have $s = \exp(0, d_0)$, which is the pivot program.
Now, once we receive the next d_i as input, we check whether d_i is equal to $2^{\exp(0,d_i)}3^{\exp(1,d_i)}$. If not, we know that the input cannot belong to a set in the class, so we may as well as stop here and return 0.
If we do have the sought form, we ask whether or not the corresponding program s_i matches the program s. If not, we again are surely outside the class, so we return 0.
Otherwise, we compute $t = [s_i \in K]_{t_i}$ and then proceed depending on the outcome.

1. $t < t_i$. In this case, the amount of steps extends the minimal amount, so we do not have a subset of some $M_{x,y}$, rather a superset. So, we are surely not in the class and thus return 0.

[3] This is to be seen as a code. We define the new value of b as either $b + 1$ or t, depending on which one is bigger.

2. $t = t_i$. In this case, we have a candidate for a class member. So, we output a program of the set $\{d_0, d_1, \ldots, d_i\}$, i.e. the set of input that we have gathered so far.
3. $t = t_i + 1$. When we get to this case, we may have one of the two following situations. We may have a candidate, namely a proper subset of some $M_{x,y}$, or we may have a program that actually is never enumerated into K. Here, the counter comes to use. We set the counter b to the maximal value of the counters last value plus one, i.e. $b + 1$, or t. Then we ask, whether or not the program was enumerated into K in b steps. By doing so, we make sure to check the next, yet unchecked, step, to see whether or not we enumerate s into K in some later step, namely b. If so, we can output the program of $\{d_0, \ldots, d_i\}$, as we may have hit some proper subset of some $M_{x,y}$. In the other case, we output 0 and request the next input. By doing so, we ensure the machine to converge if the input does not belong to a set of the class.

We see that the algorithm works properly. So, the class is confidently learnable.

Addition (for Example 13). We will show that $\mathcal{C}_n$ is CI. For some input $(d_n)_n$, the following algorithm will do the trick. Let p_0 be the program of the zero function. Then,

1. given d_i and p_i, check whether $\varphi_{p_i}(0) < n$ and if so, do
   (a) if $\varphi_{p_i}(d_i + 1) = 0$, then p_{i+1} is a program of

$$f(x) = \varphi_{p_i}(x) + \overline{\operatorname{sgn}}(x) + \overline{\operatorname{sgn}}(|x - (d_i + 1)|),$$

   (b) else, $p_{i+1} = p_i$,
2. else, $p_{i+1} = p_i$.

The idea is simple. Position 0 of the program counts the different data that occurred so far. Once it reaches n, i.e. case 1 is not true anymore, the computation will converge, as either we have met some set in the class correctly, or we expanded it. Otherwise, while we do not have all the elements, in case 1a we check whether the datum is really new, i.e. $\varphi_{p_i}(d_i + 1) = 0$. If so, we add one to the counter, i.e. $f(0) = \varphi_{p_i}(0) + 1$, and mark the element as seen, i.e. $f(d_i + 1) = 1$. Then the algorithm outputs a code of this current characteristic function. Since this case can only occur finitely often, the algorithm will converge here, too.
In the end, we can rebuild the characteristic function of $S_{\mathcal{C}_n}$ via $\chi_{S_{\mathcal{C}_n}}(x) = \varphi_p(x + 1)$.

Addition (for Remark 20). Let $\mathcal{U} \in$ vCI be a class with an infinite set $U = \{u_0, u_1, \ldots\}$. Notice that confident classes cannot contain any infinite ascending chain. So, $\forall_n^\infty m : \{u_0, \ldots, u_m\} \notin \mathcal{U}$. Then, $\forall_n^\infty i : \mathtt{vc}(u_0, \ldots, u_i) = -1$, however, $\mathtt{vc}(.)$ should converge to some code of U.

References

1. Angluin, D.: *Inductive inference of formal languages from positive data.* Information and Control 45 (**1980**), 117-135. doi.org/10.1016/S0019-9958(80)90285-5
2. Cooper, S. B.: *Computability Theory.* Chapman & Hall, **2004**.
3. Gao, Z., Stephan, F.: *Confident and consistent partial learning of recursive functions.* Theoretical Computer Science 558 (**2014**), 5-17.
4. Harizanov, V. S.: *Inductive Inference Systems for Learning Classes of Algorithmically Generated Sets and Structures.* Induction, Algorithmic Learning Theory, and Philosophy (**2007**), 27-54.
5. Jain, S.: *Hypothesis spaces for learning.* Information and Computation 209 (**2011**), 513-527. doi.org/10.1016/j.ic.2010.11.016
6. Jain, S., Lange, S., Zilles, S.: *Consistent and conservative iterative learning.* Technical Report. **2007**.
7. Jantke, K. P.: *Reflecting and self-confident inductive inference machines.* Algorithmic Learning Theory 1995 (**1995**), 282-297.
8. Lange, S., Zeugmann, T.: *Incremental learning from positive data.* Journal of Computer and System Sciences 53 (**1996**), 88-103.
9. Lange, S., Zeugmann, T.: *Language Learning in Dependence on the Space of Hypotheses.*
10. Lange, S., Zeugmann, T.: *Learning recursive languages with bounded mind changes.* International Journal of Foundations of Computer Science (**1993**), 157-178.
11. Rogers, Jr., H.: *Theory of Recursive Functions and Effective Computability.* First MIT paperback edition, third printing. **1992**.
12. Stephan, F.: Script on *Einführung in die Lerntheorie.* **2002/2003**.