Utrecht University

Learning paradigms classified by the arithmetical complexity of their learnable language families

by

Yfke Dulek
student no. 3471098

A 7.5 EC thesis submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science

in the
School of Philosophy and Artificial Intelligence
Faculty of Humanities

Supervisors:
Dr. Jaap van Oosten
Dr. Rosalie Iemhoff

Date:
July 10, 2013

Contents

Introduction **1**

1 Recursive functions **3**
- 1.1 Recursive functions: partial and primitive . . . 3
- 1.2 Indices for Turing machines, functions and sets . . . 6
- 1.3 Some recursive functions: pairing and coding . . . 7

2 Learning theory **9**
- 2.1 Gold's model . . . 9
- 2.2 Relation to human language acquisition . . . 10
- 2.3 Variations on Gold's model . . . 12

3 The arithmetical hierarchy **13**
- 3.1 Definition . . . 13
- 3.2 Many-one reduction . . . 15

4 Classifications **17**
- 4.1 Explanatory learning . . . 18
  - 4.1.1 Upper bound: EXL $\in \Sigma_4$ . . . 18
  - 4.1.2 Lower bound: EXL is Σ_4-hard . . . 20
- 4.2 Finite learning . . . 27
  - 4.2.1 Upper bound: FINL $\in \Sigma_3$ . . . 27
  - 4.2.2 Lower bound: FINL is Σ_3-hard . . . 28
- 4.3 Behaviorally correct learning . . . 31
  - 4.3.1 Upper bound: BCL $\in \Sigma_5$ . . . 31
  - 4.3.2 Lower bound: BCL is Σ_5-hard . . . 32
- 4.4 Anomalous learning . . . 38
  - 4.4.1 Upper bound: EXL* $\in \Sigma_5$ . . . 38
  - 4.4.2 Lower bound: EXL* is Σ_5-hard . . . 38

5 Conclusion **41**

Appendices **44**

A Prenex normal forms of formulas **44**
- A.1 The description of EXL is Σ_4 . . . 45
- A.2 The description of FINL is Σ_3 . . . 47
- A.3 The description of BCL is Σ_5 . . . 49
- A.4 The description of EXL* is Σ_5 . . . 51

B Notations used **53**

Introduction

Machine learning is one of the core subjects in artificial intelligence. Since the dawn of the computer era, many researchers have tried not only to devise efficient and effective machine learning algorithms, but also to explore computational boundaries. Can a machine learn anything? Can it eventually become indistinguishable from a human being in its capabilities? But also: can the limits of what can be taught to a computer shed light on the limits of human cognition?

One particular skill that separates humans from other organisms is our remarkable ability to learn and use languages. A child receives very incomplete data about the language it is learning. It is hardly ever explicitly told what grammatical rules underlie the language, or which sentences are correct and which are not [8]. It has to learn the language based solely on some positive examples, and even this data can be very noisy. But somehow this does not matter: eventually the child is able to produce correct *new* sentences of the language, even though it has been presented with a limited amount of examples of sentences.

In the light of this phenomenon, algorithmic learning theory was developed by Gold [6] in 1967. Roughly, Gold's abstract model of language learning comes down to the following: a machine is supplied with a stream of positive instances of some (formal) language, one at a time, possibly with repetition, but in such a way that every grammatical sentence in that language will be presented to the machine at some point. Every time it is being presented with an instance, the machine replies with a guess about the nature of the language. Initially, these guesses will probably be incorrect, but as the machine is presented with an increasing number of examples, it may gather enough information about the language to infer its exact contents. If, from some point on, the machine's guess is correct and never changes any more, the machine is said to have *identified* the language 'in the limit' from this particular stream of positive instances. A machine is said to *learn* a language if it can identify it from any possible data stream, regardless of the order in which those instances are presented. A language family, which is simply a set of languages, is *learnable* if there exists a single machine that is able to learn all languages in that family.

In the past years, some natural variations on the identification criterion defined by Gold have been developed [4]. For example, one could require from the machine not only that it converges to a correct guess after some time, but also that it knows *when* it has done so. This is a stronger requirement, and one may expect that some language families that were learnable under Gold's criterion may not be under this new criterion. On the other hand, one could be more lenient and allow the machine to make small mistakes: as long as the language it settles on is close enough to the language being presented, the machine's answer could still be considered correct. As one might expect, the number of learnable language families increases as a result of such a modification.

In Beros [2], four different identification criteria are compared by classifying the class of language families learnable under these criteria in the so-called arithmetical hierarchy: if such a class falls into a higher category in this hierarchy, Beros claims, this is a sign that those language families are more complex and therefore the learning process more sophisticated. Thus, his classifications allow for a way of objectively comparing the strength of several different learning paradigms.

This work aims to provide an overview of and insight into Gold's learning theory and the proofs of Beros' classifications. Recursion theory and learning theory are introduced on a level suitable for those unfamiliar with the field. For a more extensive treatment, the reader is referred to the works of Soare [13] and Osherson et al. [10]. The proofs of Beros's recent paper are explained at a level that is accessible to undergraduate students, thereby providing a self-sustained account of some of the progress currently made in the field of learning theory.

Gold's model as described above will be treated more thoroughly in Section 2, and some limitations and variations on this model will be discussed as well. To this end, several basic concepts of recursion theory will be dealt with first, in Section 1. Recursive functions are exactly those functions that are computable by Turing machines [7]. It may sometimes be useful to change perspective from Turing machines to recursive functions or vice versa in order to obtain certain results, so these notions will be used interchangeably.

In Section 3, the arithmetical hierarchy will be defined. The key relation which this hierarchy is built upon is that of m-reducibility: a relation between two sets which in effect states that the decision problem for the first set (i.e. deciding whether a certain element belongs to the set or not) is no harder than the decision problem for the second. This relation will also be defined in Section 3, and some basic properties of the arithmetical hierarchy will be stated.

Section 4, the main body of this work, will be concerned with dissecting the classifications in Beros [2]. First, the proof of the classification of the class of families learnable under Gold's original identification criterion, TxtEx, will be treated in much detail. Then, for three variations (TxtFin, TxtBC, and TxtEx*), there will be a stronger focus on the general concepts and structures, leaving out the justification of several minor details. After having read the proof of the classification of TxtEx, the reader is trusted to be able to judge these details to be provable.

Finally, in section 5, the results will be summarized and their significance and implications will be discussed.

1 Recursive functions

The purpose of this section is to provide a compact introduction to the concepts of recursion theory that are relevant to this work. First, the concept of a recursive function and its connection to Turing machines will be described. Second, a way of indexing Turing machines, functions and sets is defined. Third, some important recursive functions related to the coding of tuples as single numbers are treated.

1.1 Recursive functions: partial and primitive

Recursion theory is concerned with functions on the natural numbers that are *effectively calculable*: that is, functions that can be calculated in an 'algorithmic' manner. Although recursive functions were originally developed separately from the notion of Turing machines, the two have been shown to be equivalent in computing power by Kleene [7]; so recursive functions may be thought of as functions for which there exists a Turing machine that calculates exactly that function. In this light, it is useful to mention that recursive functions need not be total: their domain can be a subset of the natural numbers – exactly those numbers on which the associated Turing machine eventually halts and returns an output. If the Turing machine never halts on a given input, the function does not have a value for that number. This is why these functions are often spoken of as *partial* recursive functions.

In the notation of recursive functions, λ-notation has been adopted in order to avoid any confusion about which letters denote variables and which denote fixed values. If a λ is followed by a series of lower case letters, this indicates that these letters are variables that have yet to receive their values. For example, $\lambda xy.(x+y)$ is a function of two variables, x and y, that returns the sum of two input values, while $\lambda x.(x+y)$ denotes a single-variable function that adds to its input a fixed value, y. λ-notation is also useful to describe functions without having to explicitly name them, as in clause (2) of Definition 1.

The next definition of partial recursive functions is based on Soare [13].

Definition 1. *The class of partial recursive functions $\mathcal{P}$ is the smallest class of functions such that:*

1. *The successor function $S = \lambda x.(x+1)$ is in $\mathcal{P}$.*
2. *The constant functions $\lambda x_1 \cdots x_n.m$ are in $\mathcal{P}$ for all $n, m \geq 0$.*
3. *The projection functions $\pi_i = \lambda x_1 \cdots x_n.x_i$ are in $\mathcal{P}$ for all $n \geq 1$ and $1 \leq i \leq n$.*

4. *$\mathcal{P}$ is closed under* ***composition****: if the functions $g_1, ..., g_m$ are functions of n variables, h is a function of m variables, and all are in $\mathcal{P}$, then so is $f = \lambda x_1 \dots x_n.h(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$.*

5. *$\mathcal{P}$ is closed under* ***primitive recursion****: if g is a function of $n-1$ variables and h is a function of $n+1$ variables and both are in $\mathcal{P}$, then so is f, defined by*

$$\begin{aligned} f(0, x_2, ..., x_n) &= g(x_2, ..., x_n) \\ f(x_1+1, x_2, ..., x_n) &= h(x_1, f(x_1, ..., x_n), x_2, ..., x_n) \end{aligned}$$

6. *$\mathcal{P}$ is closed under* ***unbounded search*** *or* ***minimalization****: if g is a function of $n+1$ variables and g is in $\mathcal{P}$, then so is f, defined by*

$$f(x_1, ..., x_n) = \mu y.(g(y, x_1, ..., x_n) = 0)$$

(provided that for all $x \leq y$, $g(x, x_1, ..., x_n)$ is defined). Here, μy denotes "the least y such that".

For example, the function $f(x_1, x_2) = x_1 + x_2$ is recursive, since it can be defined with use of the primitive recursion rule, composition rule and the successor and projection functions:

$$\begin{aligned} f(0, x_2) &= 0 + x_2 = x_2 = \pi_1(x_2) \\ f(x_1+1, x_2) &= x_1 + 1 + x_2 = S(x_1 + x_2) = S(\pi_2(x_1, f(x_1, x_2), x_2)) \end{aligned}$$

Here, the functions g and h as mentioned in clause (5) of Definition 1 are π_1 and $S \circ \pi_2$, respectively. S and π_2 are composed using clause (4). The recursiveness of f should not come as a surprise: one can envision the task of addition as an algorithmic one.

To strengthen the correspondence between partial recursive functions and Turing machines, let us convince ourselves that partial recursive functions as described in Definition 1 can indeed all be computed in an algorithmic way (that is, by a Turing machine). The functions described in the first three clauses are fairly straightforward to implement as a Turing program: for example, a Turing machine computing a constant function $\lambda x_1 \cdots x_n.m$ simply discards all input and returns m. For the last three clauses, assume that Turing machines for the functions g, g_i and h already exist. To calculate the function f of clause (4), a Turing machine should copy the input m times, run the Turing programs for g_1 up to g_m on those inputs, store the results and feed them to a Turing program for h. Provided that all these programs halt on the inputs they receive, the composed Turing machine for f should not present any problems either. A similar observation can be made for clause (5) of the definition: a Turing machine for f can start out by calling the program for g and then feed the output to a program for h a number of

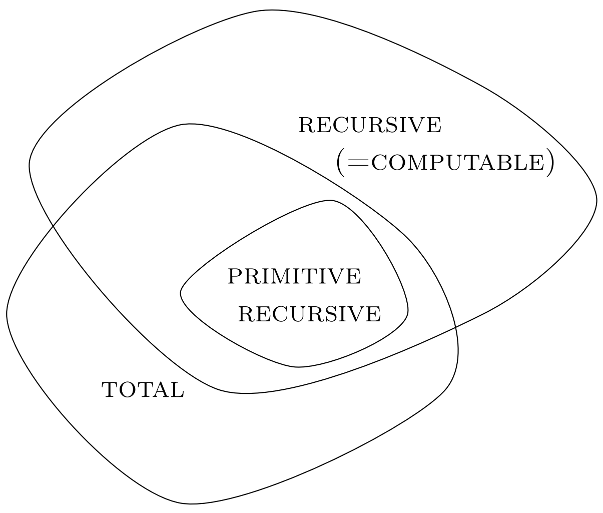


Figure 1: *A Venn diagram illustrating the relation between recursive, total and primitive recursive functions. The domain is that of all partial functions: total functions are just special cases of partial functions.*

times until the desired value of x_1 has been reached. Again, if the Turing machines for g and h halt on all inputs that are fed to them during this computation, this should pose no problems.

However, the case for unbounded search in clause (6) is slightly different. A Turing machine for f searches for the least y such that the Turing machine for g returns 0 on the input $(y, x_1, ..., x_n)$. It is possible that the machine continues searching forever, never finding an appropriate value for y. It can never conclude that such a y does not exist, since it might just be a really large number. So, if no appropriate y exists, the Turing machine does not halt and the function f is undefined on $(x_1, ..., x_n)$. This is denoted $f(x_1, ..., x_n) \uparrow$ (if the function were defined, the notation would be $f(x_1, ..., x_n) \downarrow$).

Recursive functions may be partial if somewhere in their definition unbounded search occurs. The class of functions that are defined using only clauses (1)-(5) of Definition 1 is the class of *primitive recursive functions*. The addition function $\lambda xy.(x + y)$ is an example of such a primitive recursive function. In fact, a lot of standard functions (e.g. multiplication, exponentiation) are primitive recursive. Because they cannot employ unbounded search, all primitive recursive functions can be guaranteed to be total. Not all total recursive functions are necessarily primitive, however: functions defined by unbounded search might coincidentally be total. See also Figure 1.

Finally, not only functions, but also sets and predicates can be labelled (primitive) recursive. A set A is (primitive) recursive if its *characteris-*

tic function[1] is. An n-place predicate P is (primitive) recursive if the set $\{(x_1, ..., x_n) \mid P(x_1, ..., x_n)\}$ is. An important primitive recursive predicate, Kleene's T-predicate, will be discussed in Section 1.2.

1.2 Indices for Turing machines, functions and sets

A Turing machine can be described by listing all its characteristics: its states, tape alphabet, transition rules, et cetera. While such a description can become tediously long, it is always finite and hence the string that describes the machine can be coded into a natural number in some computable way [12]. The notation M_e refers to the Turing machine with code (or *index*) e.

The same index can also refer to a function: ϕ_e denotes the recursive function that M_e computes. However, different Turing machines can effectively compute the same function. The codes of those machines are then both indices for the same function. In fact, the Padding Lemma states that any recursive function has a countably infinite number of different indices [13].

Extending the concept of indices even further, W_e denotes the domain of the function ϕ_e. Any set that has such an index (i.e. is the domain of some recursive function) is called *recurively enumerable* (r.e. for short) or, in the terminology of Turing machines, *Turing recognizable.* W_e is the set of those inputs on which the machine M_e halts.

The indexing described above not only provides a straightforward way of identifying functions, machines and sets, it also gives rise to a key predicate in recursion theory: the primitive recursive *Kleene T-predicate* [7]. The four-place predicate $T(m, e, x, y)$ is true if and only if e is an index for a Turing machine, and this machine M_e, upon receiving the m-tuple x as its input, performs the (halting) computation y, which is a list of all steps the machine has performed after receiving the input x. The input tuple and the computation are not natural numbers themselves, but are coded into a natural number in such a way that the contents can be obtained from the codes in a primitive recursive manner. The details of this coding are explained in the next subsection. From the code y, the function $\lambda y.U(y)$ primitive recursively recovers the output of the computation.

The Smn-theorem [13] provides a way to computably determine the index of a function. This theorem states that, given the index e of a recursive function $\phi_e = \lambda x_1 x_2 \cdots x_m y_1 y_2 \cdots y_n.f(\vec{x}, \vec{y})$ and the values $\vec{a} = a_1, a_2, ..., a_m$, an index for the function $\lambda y_1 y_2 \cdots y_n.f(\vec{a}, \vec{y})$ can be found primitive recursively. In other words, it allows us to find the index for the function ϕ_e with certain inputs fixed or parametrized. For this reason, the Smn-theorem is also sometimes called the Parameter Theorem.

[1]The function χ_A such that $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ otherwise

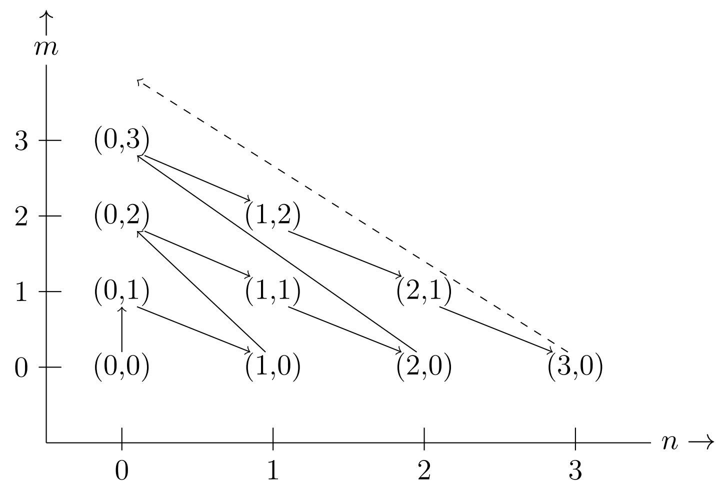


Figure 2: The function $j(n, m) = \frac{1}{2}(m + n)(m + n + 1) + n$

1.3 Some recursive functions: pairing and coding

In this subsection, some recursive functions will be treated that will be used throughout this work. A lot of 'standard' functions are recursive. Addition, cut-off subtraction (a function that returns 0 if the outcome would be negative), multiplication, division (although rounded down to an integer), exponentiation, maximum, minimum, and remainder functions are all examples of recursive functions. As mentioned in the previous subsection, tuples of numbers can also be coded in such a way that the original tuple can be recovered primitive recursively. The functions that deal with encoding and decoding of tuples or sequences are explicated here.

To start, there is the *pairing function* j. This primitive recursive function provides the well-known bijection from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ as pictured in Figure 2. j codes pairs, but its functionality can be extended to tuples of any kind. The functions j^i, that code i-tuples, are constructed from j as follows:

$$\begin{array}{rcl} j^1(x_1) & = & x_1 \\ j^2(x_1, x_2) & = & j(x_1, j^1(x_2)) \\ j^3(x_1, x_2, x_3) & = & j(x_1, j^2(x_2, x_3)) \\ \vdots & & \vdots \\ j^n(x_1, ..., x_n) & = & j(x_1, j^{n-1}(x_2, ..., x_n)) \end{array}$$

Note that these functions j^i are all bijections. So, every natural number is a code for a unit, a pair, a triple, et cetera. Every element of every such a tuple can be recovered by the primitive recursive 'inverses' j^i_m (with $0 \leq m < i$), that return the mth element of the i-tuple coded by their input.

To decode a natural number, information is needed about the length of the tuple it is supposed to code, in order to know which inverse function to call. To bypass this, a new coding function $\langle\cdot\rangle$ is constructed. Instead of coding several different tuples of different lengths, every tuple will have its own unique code. This coding starts at the empty sequence ϵ, the only tuple of length 0. For other tuples, the length n of the tuple is coded along with the code $j^n(\vec{x})$. The information about the length is now stored in the code itself. Some work has to be done (e.g. using $n-1$ instead of n) to ensure that the function $\langle\cdot\rangle$ is a bijection.

$$\begin{aligned}\langle\epsilon\rangle &= 0\\ \langle x_1,...,x_n\rangle &= j(n-1, j^n(x_1,...,x_n))+1\end{aligned}$$

Since the number of arguments for this coding function is not fixed, one cannot speak of the recursiveness of it. There are however some important primitive recursive functions that act on codes of tuples:

- Decoding: $\lambda xi.(x)_i$ returns the i^{th} element of the sequence coded by x, provided that that sequence is long enough to have an i^{th} element. Otherwise, it simply returns 0.
- Length: $\lambda x.\text{lh}(x)$ returns the number of elements of the sequence coded by x.
- Concatenation: $\lambda xy.x\star y$ concatenates the two sequences coded by x and y and returns the code of that concatenation.

Codes can be used for a variety of purposes. For example, entire computation logs of Turing machines can be stored into a single (albeit fairly large) number, because a computation is simply a list (tuple) of all steps the machine performs. Such a step is itself a list containing information on the state, tape contents and tape head position of the machine. The coding of such a computation log is useful for the Kleene T-predicate. In general, coding is used to compress information into a single number, in order to feed it to a recursive function or Turing machine.

2 Learning theory

The general concept of learning theory was described in the introduction, but with the tools of recursion theory it can now be outlined in more detail. Learning theory was first developed by Gold [6] from a linguistic point of view. It was intended to mimic human language acquisition, but this goal is obstructed by two factors: firstly, the process of language acquisition is very complex and simplification is unavoidable for its modelling. Secondly, research on human language acquisition is still ongoing and a lot of gaps are still to be filled in. Some of the differences between Gold's model and human language acquisition have led to the development of variations on the model. These difference and the resulting learning models are discussed in Sections 2.2 and 2.3.

2.1 Gold's model

Intuitively, a *language* can be described as the set of grammatical sentences in that language. In order to fit the learning model to recursive functions, however, languages are taken to be subsets of $\mathbb{N}$. To see why this is a reasonable abstraction, fix some finite alphabet and consider the set of strings in this alphabet, including all nonsensical strings. This set can be listed simply by first listing all strings of length 0, followed by all strings of length 1, et cetera. This creates a bijection between $\mathbb{N}$ and the set of all strings in the alphabet. The language is now the set of those numbers that correspond to grammatical sentences.

As mentioned before, a *learner* is simply a Turing machine or, equivalently, a recursive function. At every point in time t, the learner is presented with an example, a single element of the language L it is learning. Which elements are presented at which points in time is determined by a total (not necessarily recursive) function f, the range of which should exactly equal L. Since a Turing machine does not have any kind of permanent storage allowing it to recall previous inputs, it will not only receive the current example $f(t)$, but the whole list $(f(0), f(1), ..., f(t))$, coded into a single input number by the function $\langle\cdot\rangle$ from section 1.3. The code $\langle f(0), f(1), ..., f(t)\rangle$ contains of the first $t+1$ elements presented by f and is denoted $f \upharpoonright (t+1)$.

A function f with range L is called a *text* for L. It provides the learner with information about L in such a way that the learner only receives positive instances of the language it is learning. Moreover, every element of L will be presented to the learner at some point in time. In principle, no restrictions are put on f other than that it should be total. However, in some variations of the standard learning model, one could require texts to be recursive or have certain other properties. In the classification proofs, we will encounter such extra demands several times.

In response to the information it receives on a point in time t, the learner returns a *hypothesis* about the nature of the language L. This hypothesis is a number e that is interpreted as representing the set W_e (the domain of the function with index e). Note that because of this, the learner can only hypothesize about r.e. languages. Luckily, this is a wide enough range of languages for the purposes of studying natural language. Natural languages are generally accepted to fall into the category of mildly context-sensitive languages [1], and all (mildly) context-sensitive languages are recursively enumerable [9].

A learner is said to *identify* the language L from a text f if it converges toward a single correct hypothesis. From some point on, the learner needs to always output the same index e and that index should be correct (that is, $W_e = L$). Ideally, a learner is able to identify L regardless of what specific text it is presented with, as long as that text is a function with range L. If this is the case, the learner is said to *learn* L.

Note that for any language $L = W_e$, there exists a learner that learns it: the constant function that always outputs e will have instantly learned the language. Things only become interesting when whole families (i.e. sets) of languages are considered: one can wonder whether there exists a single learner that is capable of learning all the languages in the family (for example, does, in theory, a single machine exist that can learn all natural languages?). If such a learner exists, the language family is *learnable*. A basic observation can be made about the learnability of language families. If some language family $\mathcal{L}$ is learnable, then any subset of $\mathcal{L}$ is learnable as well. In fact, it is learned by the same learner. However, the converse of this implication is that if some language family $\mathcal{L}$ is *not* learnable, then no superset of $\mathcal{L}$ can be learnable.

2.2 Relation to human language acquisition

Gold's model is a very abstracted form of learning. This subsection will investigate some of the discrepancies between language acquisition in Gold's model and human language acquisition. In section 2.3 three variations on Gold's model are discussed that deal with some of these differences.

First of all, in learning theory, any family containing a single (r.e.) language is learnable by a constant function that always returns the index for that language. This exposes the fact that in Gold's model, the learner is in no way required to exhibit any kind of the 'intelligent' behaviour usually associated with learning, especially that of complex languages. Only when trying to learn a family containing an infinite amount of languages, some tactic is required beyond keeping a (finite) list of those elements that make each language unique. It can be argued that, although only finitely many natural languages actually exist at one point, a human is capable of learning practically infinitely many variations on them (for example, replace the word ‘table’ in English by ‘vable’, and a language results that could be learnt just as easily as any other variation of English in use today). So, the fact that humans seem to employ an intelligent learning strategy may just be due to the fact that they have to be extremely versatile and able to learn a vast (nearly infinite) amount of languages.

Next, learning theory assumes that a learner converges toward a perfectly stable and correct representation of the language it is learning. In reality, however, no single person has such a perfect representation of his own native language. Some words may be unknown to him, or some grammatical rules may not be obeyed. We would still like to say, however, that he has mastered his own native language. Within the framework of learning theory, this difference can be bridged only partly. The model TxtEx* discussed in section 2.3 attempts to do so.

The fact that children settle on an imperfect language may be partly due to them being presented with noisy and incomplete data about the language to be learned. If a certain word is not part of the vocabulary of people in the environment of a child, the child will probably never adopt that word into its own vocabulary. Conversely, if the child is consistently presented with a certain ungrammatical construction, it may imitate this in its own sentences. However, most of the noise in the data children are presented with is filtered out quite easily. In Gold’s model, the learner is required to hypothesize languages that are fully consistent with the presented data, so noise cannot be filtered out.

Furthermore, Gold’s learning criterion is intensional: the learner needs to settle on a single index for a language that has infinitely many indices. So it is the *grammar* it has to settle on, not the set that that grammar generates. This does not reflect the way in which humans learn. A human may change the mental representation of a language (e.g. in order to represent it more efficiently, with less grammatical rules [10]), while the language itself remains the same. Different individuals may settle on different grammars while learning the same language. In Section 2.3, an extensional learning criterion will be considered that focuses on the contents of the language itself instead of the underlying structure imposed on it by a Turing machine or recursive function.

Moreover, Gold focuses entirely on the lexical data the child receives. No attention is paid to other, non-linguistic factors that may affect the learning function of a child. There are a host of factors that may play a role, such as a child’s mood or the amount of sunlight or affection it has received that day [10]. There is no way of telling which factors are of influence here. Further research into this would be necessary first.

Finally, the learnability question for a language family is only interesting if that language family contains infinite languages. This is because the family of all finite languages is learnable by a machine that always hypothesizes the set of those numbers it has been presented with. So for Gold’s model to be an interesting (abstract) model of human language acquisition, one would have to assume that at least some natural languages are infinite. While this is generally assumed, it has not yet been thoroughly validated [11].

2.3 Variations on Gold's model

This work will be concerned with four different learning models in total. These models all adopt the same framework as Gold in terms of the definitions of learner, texts and hypotheses. The only point on which the models differ is their *criterion of success*: when can a learner be said to have successfully learned a language?

TxtEx or explanatory learning is the name for Gold's original model. A learner is required to eventually converge to a single, correct hypothesis.

TxtFin or finite learning requires the learner not only to converge toward a single, correct hypothesis, but also to *know* at any point whether it has converged yet or not. A learner should output a special symbol `?` to indicate that it is not yet sure about the nature of the language and is still awaiting additional information. Once it has received enough examples, it is to output a 'real' hypothesis and the first such hypothesis should immediately be correct. This way, it incorporates some form of self-awareness in the learning process. While TxtFin-learning is realistic in the sense that human learners do not immediately start to hypothesize about the language they are being presented with, it fails to simulate the way a child first adopts approximations of the actual language before it masters the language itself, and is never really aware of the exact moment it is 'done' learning, if ever.

TxtBC or behaviourally correct learning adopts an extensional criterion of success. The learner, like in TxtEx-learning, should in the limit settle on the language it is presented with. However, it is allowed to alternate between different indices (representations) for that set, while a TxtEx-learner should stick with a single index or name for the language.

Finally, TxtEx* or anomalous learning reflects the fact that a learner may not settle on a *perfect* representation of the language. For two languages L and L', the symmetric difference $L \triangle L'$ between these languages is the set $(L \setminus L') \cup (L' \setminus L)$. In TxtEx*-learning, the learner needs to settle on a single index like in TxtEx-learning, but the language associated with this index is allowed have a *finite* symmetric difference with the actual language it is presented with. However, this insufficiently reflects how humans settle on imperfect languages. If a person misses one word in its vocabulary, then immediately, an infinite amount of sentences is no longer at his disposal. In that case, the symmetric difference of the language it is learning and the language it settles on is infinite. However, allowing an infinite (or even just recursive) symmetric difference would allow the learner to simply settle on the empty set instead of on the language it is presented with.

3 The arithmetical hierarchy

In Section 2, the learning models TxtEx, TxtFin, TxtBC and TxtEx* have been introduced. These four models differ in learning power: some language families that are learnable in one model may not be in another. In his paper, Beros tries to “establish a measure of the complexity of the learning process” [2] of a model by examining the class of all language families that are learnable in that model. If that class is complex in its structure, this indicates that the learnable language families are very specific, i.e. deciding whether or not a given language family is learnable is difficult and cannot be done on the basis of superficial properties alone. This suggests that the learning process involved is also more sophisticated, because it has to delve deep into the structure of the languages it is learning.

In this section, a measure to determine the complexity of these classes of learnable language families is presented. We make use of the *arithmetical hierarchy*: in this hierarchy, sets are classified on the basis of the structure of their defining formulas. The arithmetical hierarchy is precisely defined in Section 3.1. In Section 3.2, a general method for determining the location of a set in the hierarchy is discussed. This method will be used repeatedly in Section 4.

3.1 Definition

The arithmetical hierarchy consists of an infinite amount of levels, all of one of three kinds: Σ_n, Π_n or Δ_n, for $n \in \mathbb{N}$ (see Figure 3). In other literature, these sets are often denoted Σ^0_n, Π^0_n and Δ^0_n in order to stress the distinction from other systems such as the analytical hierarchy. Since this work is only concerned with the arithmetical hierarchy, the superscript 0 will be omitted.

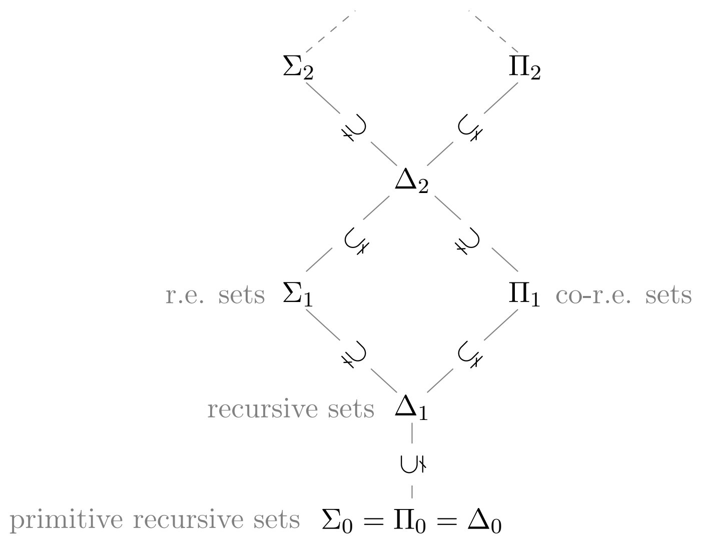
First, the sets Σ_n, Π_n and Δ_n are defined in Definition 2. Then, the structure of the hierarchy and some of its properties will be elaborated upon.

Definition 2. *The levels Σ_n and Π_n are simultaneously defined by induction as follows [14]:*

- *$A \in \Sigma_0$ iff $A \in \Pi_0$ iff A is primitive recursive.*
- *$A \in \Sigma_{n+1}$ iff A is of the form $\{x \mid \exists y(x,y) \in B\}$ where $B \in \Pi_n$.*
- *$A \in \Pi_{n+1}$ iff A is of the form $\{x \mid \forall y(x,y) \in B\}$ where $B \in \Sigma_n$.*

Moreover, $\Delta_n := \Sigma_n \cap \Pi_n$ for all $n \in \mathbb{N}$.

In general, to determine the position of a set in the hierarchy, one should construct its describing formula and convert it to prenex normal form by moving all quantifiers to the front of the formula. The matrix (the quantifier-free remainder of the formula) should be primitive recursive. The number of



Σ_2
Π_2
Δ_2
r.e. sets Σ_1
Π_1 co-r.e. sets
recursive sets Δ_1
primitive recursive sets $\Sigma_0 = \Pi_0 = \Delta_0$

Figure 3: *The arithmetical hierarchy visualized. The class Σ_1 contains all r.e. sets. Π_1 consists of the complements of Σ_1 sets: these are called co-r.e. If a set is both r.e. and co-r.e., it is recursive [13].*

quantifier alternations, along with whether the first quantifier is existential or universal, determines the position in the hierarchy.

For example, the set TOT of indices e for total recursive functions may be described by the formula $\forall x \exists y T(1, e, x, y)$, which states that for any input x, there exists a (finite) computation y of the Turing machine M_e on x. This formula is already in prenex normal form. It has one quantifier alternation and starts with a universal quantifier, hence it is in Π_2. Note that immediately follows that the complement of TOT is Σ_2: indeed, $e \notin$ TOT if and only if $\neg\forall x \exists y T(1, e, x, y)$. This describing formula is equivalent to $\exists x \forall y \neg T(1, e, x, y)$, which has one quantifier alternation and starts with an existential one. This observation holds in general: if a set is in Σ_n, its complement is in Π_n, and vice versa.

Of course, the Π_3 formula $\forall x \exists y \forall z T(1, e, x, y)$ would also have been an adequate description of TOT. The set could even be proven to be in Π_{267}, but this provides a lot less information than the fact that it is in Π_2. So the challenge is to find the *lowest* level in the hierarchy to which a set belongs. One way of doing this is described in Section 3.2.

The Hierarchy Theorem states that all inclusions in the hierarchy are strict [13]: each level contains elements that are not at any lower level, ensuring that different levels of the hierarchy do not 'collapse' into one. Moreover, the levels Σ_n and Π_n, while neither is 'higher' than the other, are never equal: they both contain sets that the other does not. See Figure 3.

3.2 Many-one reduction

When classifying sets in the arithmetical hierarchy (as will be done with the classes of learnable language families in Section 4), one needs a way to prove that the classification found is indeed the 'best' classification possible, i.e. at the lowest possible level in the hierarchy. One way to prove this is by proving that the given set is at least as complex as all other sets at that level. The set can then not be at any lower level, because that would mean that all those other sets would fall into the same lower level. This would contradict the fact that the inclusions in the hierarchy are strict.

So how would one go about proving that one set is 'at least as complex' as another? For this, the concept of *many-one reducibility* or m-reducibility is used to reduce the decision problem of one set to that of another. First, decision problems and m-reducibility are formally defined.

Definition 3. *The decision problem for a set $A \subseteq \mathbb{N}$ is determining, for an arbitrary number $x \in \mathbb{N}$, whether or not $x \in A$.*

Definition 4. *Given two sets A and B, A is m-reducible to B, notation $A \leq_m B$, if there is a total recursive function f such that*

$$x \in A \Leftrightarrow f(x) \in B.$$

Intuitively, reducing a set A to B means that when deciding whether or not $x \in A$, this question can be transformed by the function f into a question about the members of B. For example, if one wants to know whether or not a number n is in the set of all even numbers, one could also ask whether the number $n + 1$ is in the set of all odd numbers. In this example, the function $f(x) = x + 1$ provides the reduction from the set of even numbers to the set of odd numbers. Note, however, that the answer about whether or not $f(x)$ is in B is itself the final answer and cannot be swapped form positive to negative or vice versa. For example, the function $f(x) = x$ does not provide a reduction from the set of even numbers to the set of odd numbers because in this case the answer to the question "Is n even?" is not the same as the answer to "Is $f(n)$ odd?". Even though knowing whether $f(n)$ is odd provides conclusive information about whether n is even, f is not a reducing function. It is not even necessarily the case that a set is m-reducible to its complement at all.

If $A \leq_m B$ and B is at some level of the hierarchy, then A is at that level as well [14]. A may also fall into lower levels of the hierarchy, but it cannot be more complex than B, meaning that its 'best' classification must be at the level of B or below that. Conversely, if the best classification of A is known, then the best classification of B cannot be any lower. These facts are useful for proving that some classification for a set X is the best or lowest classification possible. From the Hierarchy Theorem, it is known that at each level of the hierarchy, there is at least one set Y that is not at any lower level of the hierarchy. If Y is m-reduced to X, it follows that X cannot be at any lower level as well. Often, the nature of this particular set Y is not known, so a general strategy is to reduce an *arbitrary* set from that level to X. Then certainly, Y is reducible to X as well. X is then said to be *(m-)hard* at the level of the hierarchy (or Σ_n-, Π_n- or Δ_n-hard in order to emphasize the hierarchy level for which it is hard).

If a set X, in addition to being hard at some hierarchy level, is also actually at that level (in the way that Definition 2 describes), it is said to be m-complete at that level. To provide an exact classification of the four sets Beros treats in his paper, he proves that these sets are complete at a certain level of the hierarchy.

A set is 1-reducible to another if it is m-reducible and the reducing function is *injective*. The notions of 1-hardness and 1-completeness are straightforwardly similar to the notions of m-hardness and m-completeness described above. An important result used in the proof of Section 4.1.2 is the fact that a set is m-complete in Σ_n or Π_n if and only if it is 1-complete at that level [13]. Consequently, for m-complete sets the functions that reduce other sets at the same level to it can be assumed to be injective.

4 Classifications

In this section, the classifications of TxtEx, TxtFin, TxtBC and TxtEx* will be treated, in that order. The lines of Beros' proofs in [2] are followed. For each learning paradigm, the exact position of the class of learnable language families in the arithmetical hierarchy is determined. To prove that a position is exact, one not only needs to prove that the class is at a certain level in the hierarchy (and thereby providing an upper bound to its complexity), but also to prove that it is not at any simpler level. One way to do this is to prove that the class is complete in the hierarchy level – any other set of the same complexity should be able to be reduced to it.

Note that the arithmetical hierarchy is a hierarchy of sets of natural numbers, so the classes of learnable language families need to be coded into subsets of $\mathbb{N}$ by assigning 'indices' to those language families. Such an index will be the index of the total recursive function g for which the language family equals $\{W_{g(0)}, W_{g(1)}, W_{g(2)}, ...\}$. If such a function g exists, the family is said to be *uniformly recursively enumerable* [5]. Not every language family is uniformly recursively enumerable, however. This notation restricts the analysis of this section to those language families that are. The class of all learnable language families can now be regarded as the set of the indices of those families: a set of natural numbers, eligible for classification in the arithmetical hierarchy.

The following four subsections will each treat one classification proof. The proof of the classification of Gold's learning paradigm is treated first and in much detail. It has to be ensured, for example, that certain tasks can be carried out in a finite amount of time by a machine, or that an index for a constructed set can be found recursively. The techniques used to ensure this are quite general and can be applied in the other three proofs as well. So, for the last three subsections, not every step has been worked out in detail and we focus more on the general concepts.

4.1 Explanatory learning

Learning in the limit, explanatory learning, and TxtEx all are terms that denote the learning paradigm as originally defined by Gold. Though not at the simplest level of the arithmetical hierarchy, the class EXL of (codes for) language families identifiable by the TxtEx-criterion is a good place to start. Beros proves that that EXL is Σ_4-complete, and does so by showing that Σ_4 is both an upper and a lower bound to the complexity of EXL.

4.1.1 Upper bound: EXL $\in \Sigma_4$

The first thing to do is to provide a description for the set EXL in the form of a Σ_4-formula: a formula $P(e)$ that holds if and only if $e \in$ EXL. Doing so proves that the most economic classification of EXL cannot be any higher than Σ_4 and in this way provides an upper bound to the complexity of EXL.

Let $\mathcal{L}$ be a uniformly recursively enumerable language family $\{L_1, L_2, ...\}$ with index e, i.e. enumerated by the recursive function ϕ_e, as in the introduction of this section. The formula $P(e)$ will not, as one might expect, directly describe TxtEx-learnability by stating the existence of a Turing machine that learns any language in $\mathcal{L}$ from any text. Instead, it will state that there must exist a *total* Turing machine that learns any language in the family from any *recursive* text (i.e. a text described by a total recursive function instead of just a total function). This statement will turn out to be equivalent to TxtEx-learnability.

The formula $P(e)$ will have the general form

$$\exists k \forall i \Big((M_k \text{ total}) \wedge (M_k \text{ learns } L_i \text{ from recursive texts}) \Big)$$

Let us delve into the details of these two conditions:

1. M_k is a total machine (it halts on every input). This condition is introduced in order to avoid statements such as $M_k(a) \neq M_k(b)$ from being true just because $M_k(a)$ or $M_k(b)$ is undefined. It can be expressed by the Π_2-formula
$$\forall x \exists y T(1, k, x, y)$$
as seen in Section 3.1.

2. If a recursive function ϕ_a is a text for L_i – that is, it is a total function that enumerates all elements of L_i – then M_k should learn L_i from the text ϕ_a. The condition of totality of ϕ_a is analogous to that of M_k in item 1:
$$\forall x \exists y T(1, a, x, y)$$
The fact that ϕ_a enumerates all of L_i is equivalent to the statement that a number x is enumerated by ϕ_a if and only if $x \in L_i$ $(= W_{\phi_e(i)})$.

This is represented by a Π_2 formula (see Appendix A):

$$\forall x\Big(\exists y \exists z\big(T(1,a,y,z) \wedge U(z) = x\big) \leftrightarrow \exists w T(1,\phi_e(i),x,w)\Big)$$

The conclusion of this second condition, M_k learning L_i from ϕ_a, consists of two parts:

(a) M_k will eventually converge toward a final hypothesis: from some point on, its hypotheses will all be equal. The following formula A represents this fact:

$$\exists s \forall t\Big(t > s \rightarrow \Big(M_k(\phi_a \upharpoonright t) = M_k(\phi_a \upharpoonright s)\Big)\Big)$$

(b) If, at some point in time n, M_k has converged to its final hypothesis, this hypothesis should also be correct. This is exhibited by the formula B, defined as

$$\forall n\Big(\forall m\Big(m > n \rightarrow M_k(\phi_a \upharpoonright m) = M_k(\phi_a \upharpoonright n)\Big) \rightarrow W_{M_k(\phi_a \upharpoonright n)} = L_i\Big)$$

The formula as a whole,

$$P(e) = \exists k \forall i\Big((M_k \text{ total}) \wedge \Big(\big((\phi_a \text{ total}) \wedge \forall a(\phi_a \text{ enumerates } L_i)\big) \rightarrow (A \wedge B)\Big)\Big)$$

is Σ_4, as demonstrated in Appendix A. It remains to be shown that $P(e)$ holds if and only if $e \in$ EXL. This will be done by treating the two directions of the implication separately.

For the first direction, suppose that for some language family $\mathcal{L}$ with index e, e is in EXL. By definition of EXL, there exists a machine M that learns any language in $\mathcal{L}$ from arbitrary texts. While this machine is not necessarily total, its existence does ensure the existence of a total machine N that learns the family coded by e in the following way.

$$N(\langle x_0, x_1, ..., x_n\rangle) := \begin{cases} M(\langle x_0, x_1, ..., x_a\rangle) & \text{for the largest } a \leq n \\ & \text{such that } M \text{ halts within} \\ & n \text{ steps of the computation} \\ 0 & \text{if such } a \text{ does not exist} \end{cases}$$

N only needs to simulate finitely many of M's computation steps, so it always halts. To verify that N indeed learns any language that M can learn, fix a language L_i and a text for that language, and let σ be the smallest initial segment of that text for which M has entered the convergence state, i.e. $M(\sigma)$ is correct and M will stick to its answer. If N is presented with

σ, it may not be able to simulate all of M's computation steps and might settle on one of M's premature hypotheses, or output 0. However, later on, as the lengths of the inputs increase, N *will* be able to simulate the entire computation. It will then either output $M(\sigma)$ or some later hypothesis of M on this text – but these are all equal. So the total machine N, while probably not as quickly as M, will eventually converge toward the (correct) hypothesis $M(\sigma)$ and is therefore able to learn the language L_i from any text for that language, since M is too. Then certainly, it is also able to learn it from a recursive text. The criteria formulated in $P(e)$ are met and the formula holds for e.

For the other direction, suppose that $P(e)$ holds for some index e. Then we know from the formula that there exists a total machine that learns the family from recursive texts, while Gold demands that the Turing machine recognize the languages from *arbitary* texts. However, Blum and Blum [3] showed that if a family of languages can be TxtEx-learned from recursive texts, then it can also be TxtEx-learned from arbitrary texts by another machine. The machine that learns from arbitrary texts is not necessarily total, but this is not a requirement for TxtEx-learning. So the language family indexed by e is indeed TxtEx-learnable, hence $e \in$ EXL.

From the above it can be concluded that the formula $P(e)$ accurately describes the set EXL. Hence, EXL $\in \Sigma_4$, and Σ_4 is an upper bound to the complexity of EXL.

4.1.2 Lower bound: EXL is Σ_4-hard

In this subsection, a lower bound to the complexity of EXL is provided by showing that deciding EXL is at least as hard as deciding any other set in Σ_4: any Σ_4-set P is m-reducible to EXL. Fix an arbitrary Σ_4-set P. The goal for this subsection is to construct a family $\mathcal{H}_e$, dependent on e, such that $e \in P$ if and only if $\mathcal{H}_e$ is TxtEx-learnable (i.e., its index, which should be *recursively* computed from e, is in EXL). If e is not in P, $\mathcal{H}_e$ will contain a subfamily $\mathcal{F}$ that is *not* TxtEx-learnable, causing $\mathcal{H}_e$ itself to be unlearnable as well. This family $\mathcal{F}$ is fixed (does not depend on e) and is constructed first.

Construction 1. $\mathcal{F}$, a non-TxtEx-learnable family. The family $\mathcal{F}$ consists of two types of languages, H_n and L_n, defined as follows:

$$H_n = \{n + x \mid x \leq |W_n|\}$$

$$L_n = \{n + x \mid x \in \mathbb{N}\}$$

where $|W_n|$ denotes the cardinality of the set W_n. The languages in $\mathcal{F}$ are numbered, where $F_{2n} = H_n$ and $F_{2n+1} = L_n$. From n, the index of F_n

can always recursively be found using the *Smn*-theorem[2]: $\mathcal{F}$ is a uniformly recursively enumerable language family.

Note that if, for some n, $|W_n| = \infty$, then $H_n = L_n$. Otherwise, H_n will be a finite set (whereas L_n is always infinite).

The fact that $\mathcal{F}$ is not TxtEx-learnable will be proven by contradiction. Suppose that some machine M exists that *does* learn $\mathcal{F}$. From M, we will construct another machine N that, for given n, will decide in the limit whether or not $H_n = L_n$ and thus whether or not W_n is infinite. The contradiction will lie in the fact that this cannot actually be decided in the limit.

We start by constructing a (finite) sequence σ_n, dependent on n, that 'locks' M into the hypothesis L_n: on a text for L_n, after having encountered σ_n, M will never change its hypothesis again. The existence of such a *locking sequence* is proven in Blum and Blum [3]. Here, a method of recursively computing this sequence for given L_n is presented. The computation will take up several stages; $\sigma_{n,s}$ will denote the sequence σ_n as constructed up to stage s.

Stage 0 Set $\sigma_{n,0} = \langle n \rangle$.

Stage s+1 Search for a sequence τ such that $M(\sigma_{n,s}) \neq M(\sigma_{n,s} \star \tau)$. Because such a search is possibly infinite (a suitable τ may not exist), only examine those τ with $\mathrm{lh}(\tau) \leq s$ and elements in $[n, n+s]$. If a suitable τ is found, set $\sigma_{n,s+1} := \sigma_{n,s} \star \tau \star \langle n, ..., n+s \rangle$. If not, set $\sigma_{n,s+1} := \sigma_{n,s}$.

If new values of τ continue to be found infinitely often, then the infinite sequence σ_n represents a text for L_n (since at every stage, the elements n through $n+s$ are appended to σ_n). However, M does not converge toward a single hypothesis on σ_n. Infinitely often, a new extension τ of $\sigma_{n,s}$ is found for which M changes its answer. Since M should learn L_n from any text, including the text represented by σ_n, this is impossible. Thus, from some point, no extension for σ_n is ever found on which M changes its hypothesis, and σ_n remains a finite sequence. Since σ_n is part of some text for L_n, the hypothesis M outputs is a correct index for L_n.

If H_n and L_n are unequal (which is the case if W_n is finite), then M must be able to distinguish between them – so σ_n cannot be (the first part of) a text for both H_n and L_n. Conversely, if it is, then $H_n = L_n$ (and W_n is infinite). So in order to decide whether or not W_n is infinite, it suffices to know whether or not the content of σ_n is contained in H_n. We will now construct a machine N that, in the limit, is able to answer this last

[2]For example, define the function $g(n, x)$ to be 1 if $n \leq x$ and undefined otherwise. If n is odd, the *Smn*-theorem states that the index for the function $\lambda x.g(\lfloor n/2 \rfloor, x)$ with domain $L_{\lfloor n/2 \rfloor}$ can be found primitive recursively from n and the index for g. A similar function can be defined for the sets H_n.

question. On the input $\langle n, s\rangle$, N computes $\sigma_{n,s}$ and part of W_n – namely, those numbers for which the computation of ϕ_n on that number is coded by some $y \leq s$ (this partial construction of W_n is denoted $W_{n,s}$). From this, N constructs the set $H_{n,s} := \{n + x \mid x \leq |W_{n,s}|\}$. N tests whether $\sigma_{n,s}$ is contained in $H_{n,s}$ (output 1) or not (output 0). In infinity, N tests whether the contents of σ_n is contained in H_n or not. So, the set INF of those n for which W_n is infinite can be described by the following formula:

$$\exists s \forall s'(s' > s \to N(\langle n, s'\rangle) = 1)$$

This formula is Σ_2, implying that INF $\in \Sigma_2$. However, this set is known to be m-complete in Π_2 [13]. This is a contradiction[3], so it can be concluded that no machine can learn the entire family $\mathcal{F}$.

Reduction via COINF. Remember that the goal for this hardness proof is to reduce our arbitrary Σ_4-set P to EXL. This reduction will be realised in two parts: the set COINF of indices for coinfinite[4] r.e. sets will serve as an intermediate step.

First, observe that by definition of the arithmetical hierarchy,

$$P = \{e \mid \exists y((e, y) \in Q)\}$$

for some Π_3-set Q. Now, from Q, define the set Q' as follows:

$$Q' := \{(e, y) \mid \exists y' \leq y((e, y') \in Q)\}$$

Because the existential quantifier is bounded, Q' is also in Π_3. Moreover, it holds that

$$\begin{aligned} e \in P \quad &\Rightarrow \quad \exists y((e, y) \in Q) \Rightarrow \exists y \forall y' \geq y((e, y') \in Q') \\ e \notin P \quad &\Rightarrow \quad \forall y((e, y) \notin Q) \Rightarrow \forall y((e, y) \notin Q') \end{aligned}$$

The set Q', being in Π_3, can be reduced to the Π_3-complete set COINF [13]. That is, there exists a total recursive function f such that $(e, y) \in Q'$ if and only if $f(e, y) \in$ COINF. From this it follows that

$$e \in P \quad \Rightarrow \quad \exists y \forall y' \geq y(f(e, y') \in \text{COINF}) \tag{1}$$

$$e \notin P \quad \Rightarrow \quad \forall y(f(e, y) \in \text{COF}) \tag{2}$$

where COF (consisting of indices for cofinite sets) is the complement of COINF.

The structure that COINF and f have brought to P will be used in the second part of the reduction: the language family $\mathcal{H}_e$, dependent on e, is constructed such that it is learnable if and only if $e \in P$. Specifically, $\mathcal{H}_e$

[3]If INF were in Σ_2, then all Π_2-sets would be too, because they can be m-reduced to INF. This contradicts the fact that, for all n, $\Pi_n \backslash \Sigma_n$ is non-empty.

[4]A set is called coinfinite if its complement is infinite.

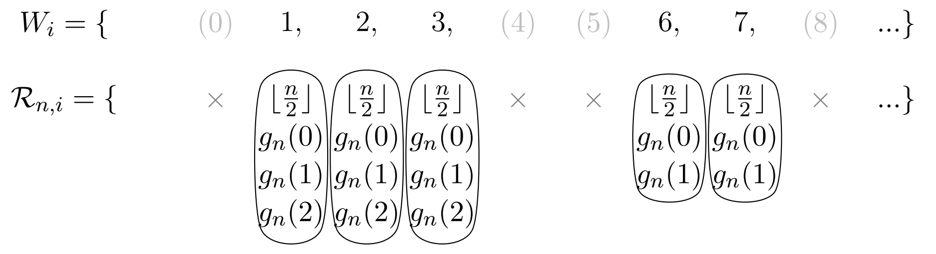

$W_i = \{$ (0) 1, 2, 3, (4) (5) 6, 7, (8) ...}
$\mathcal{R}_{n,i} = \{$ × × × × ...}
$\lfloor \frac{n}{2} \rfloor$
$g_n(0)$
$g_n(1)$
$g_n(2)$

Figure 4: *The family $\mathcal{R}_{n,i}$, where $W_i = \{1, 2, 5, 6, 7, ...\}$. For example, look at the interval $[1, 3] \subseteq W_i$: the elements $g_n(0), g_n(1)$ and $g_n(2)(= g_n(3-1))$ are enumerated into the 1st, 2nd and 3rd columns. In the current example, $\mathcal{R}_{n,i} = \{\{\lfloor \frac{n}{2} \rfloor, g_n(0), g_n(1)\},\ \ \{\lfloor \frac{n}{2} \rfloor, g_n(0), g_n(1), g_n(2)\}, ...\}$*

will be learnable if the right clause in (1) holds, and it will not be learnable if the right clause in (2) holds.

Before $\mathcal{H}_e$ is constructed, however, we present the construction of the families $\mathcal{R}_{n,i}$ for $n, i \in \mathbb{N}$. A family $\mathcal{R}_{n,i}$ will contain only finite subsets of F_n (the n^{th} language in $\mathcal{F}$) if i is an index for a coinfinite set. However, if i is an index for a cofinite set, $\mathcal{R}_{n,i}$ will contain the language F_n itself, which may be finite or infinite. Eventually, for some n and i, the families $\mathcal{R}_{n,i}$ will be used in the construction of the family $\mathcal{H}_e$. Which families are used depends on e – the details are treated in Construction 3.

Construction 2. The language families $\mathcal{R}_{n,i}$. Recall the non-TxtEx-learnable family $\mathcal{F}$ from Construction 1. Since $\mathcal{F}$ is uniformly recursively enumerable, there exists a recursive function g such that $\lambda x.g(n, x)$ (abbreviated g_n) enumerates the language F_n. The function g_n will be used to fill the languages in $\mathcal{R}_{n,i}$ in a computable way. All elements in those languages will be elements of F_n, which equals $H_{\lfloor n/2 \rfloor}$ or $L_{\lfloor n/2 \rfloor}$.

A family $\mathcal{R}_{n,i}$ can be visualized as a table, containing several columns that each represent a single language. If a column is empty, the corresponding language simply does not exist in $\mathcal{R}_{n,i}$. Eventually, the structure of $\mathcal{R}_{n,i}$ will reflect the contents of W_i: if $x \in W_i$, a language will be created in the x^{th} column of $\mathcal{R}_{n,i}$. It will always contain the number $\lfloor \frac{n}{2} \rfloor$ (the least element of F_n). If, for some $p, q \in \mathbb{N}$, $[p, q] \subseteq W_i$, then all columns $p, p+1, ..., q$ will contain the same elements, namely $g_n(0), g_n(1), ..., g_n(q-p)$ (in addition to $\lfloor \frac{n}{2} \rfloor$). For an example, see Figure 4.

If W_i is coinfinite, there will be infinitely many blank columns in $\mathcal{R}_{n,i}$: because of these 'interruptions', all columns that are filled will only receive a finite number of elements from the enumeration g_n. So, $\mathcal{R}_{n,i}$ will consist only of finite subsets of F_n.

If, however, W_i is cofinite, it contains an infinite interval $[a, \infty[$ for $a \in \mathbb{N}$.

The first $a-1$ columns of $\mathcal{R}_{n,i}$ might contain a finite number of finite subsets of F_n, but all subsequent columns in $[a, \infty[$ contain the entire enumeration g_n, and therefore equal the set F_n.

An important detail remains to be treated here. Because $\mathcal{R}_{n,i}$ is a possibly infinite family of languages and determining whether or not $x \in W_i$ is a nonrecursive task, a Turing machine cannot simply construct the entire family from the numbers n and i in a finite amount of time. It can only build it up in stages and thereby approximate the actual family. At each stage s, instead of examining the entire set W_i, the machine computes $W_{i,s}$, the set consisting of those inputs on which the machine M_i halts with a computation coded by some $y \leq s$. $W_{i,s}$ can be computed in a finite amount of time by simply going through all $y \leq s$. As s increases, more and more elements of W_i will be found, and the intervals within W_i (and therefore the contents of the columns) will grow. As s approaches infinity, $W_{i,s}$ equals W_i and the construction will equal $\mathcal{R}_{n,i}$ as defined above.

While it is impossible for a Turing machine to finitely construct the entire family $\mathcal{R}_{n,i}$, an index for a single (possibly infinite) column *can* computably be found in a finite amount of time. The function h, defined as

$$h(n,i,x,y) \simeq \begin{cases} 0 & \text{if } y \text{ is in the } x^{th} \text{ column of } \mathcal{R}_{n,i} \\ \uparrow & \text{otherwise} \end{cases}$$

is computable by a machine that simply performs the stage by stage construction of $\mathcal{R}_{n,i}$ and outputs 0 as soon as it encounters the element y in the x^{th} column. If it never does, it keeps on searching forever and the value $h(n,i,x,y)$ is undefined. By the *Smn*-theorem [13], the index of the function $\lambda y.h(n,i,x,y)$ can be found primitive recursively in n, i and x. The domain of this function is precisely the x^{th} column of $\mathcal{R}_{n,i}$, so this index is also the index for set in the xth column.

Now that we have seen that $\mathcal{R}_{n,i}$ contains the set F_n if and only if $i \in$ COF, and have established a computable way of determining an index for a single column of a set $\mathcal{R}_{n,i}$, it is time to move on to the construction of the family $\mathcal{H}_e$, which will be the union of several families $\mathcal{R}_{n,i}$.

Construction 3. The family $\mathcal{H}_e$. For the construction of $\mathcal{H}_e$, remember that the function f that reduces Q' to COINF can be assumed to be injective (see Section 3.2). Therefore, the sequence $(f(e,n))_{n\in\mathbb{N}}$ is unbounded and has an infinite, strictly increasing subsequence $(a_n)_{n\in\mathbb{N}}$. The values a_n can even be computed in a simple way: subsequently compute the values $f(e,0), f(e,1), f(e,2)$, et cetera, and count how many times a *new* maximum value is found. The n^{th} time this happens, it is the value a_n. For convenience, we would like the first value of the sequence to be 0, so define $a_{-1} := 0$.

At this point, we diverge slightly from Beros's version of the proof in order to cover every possible form of the sequence $(a_n)_{n\in\mathbb{N}}$. Define the $u(x)$

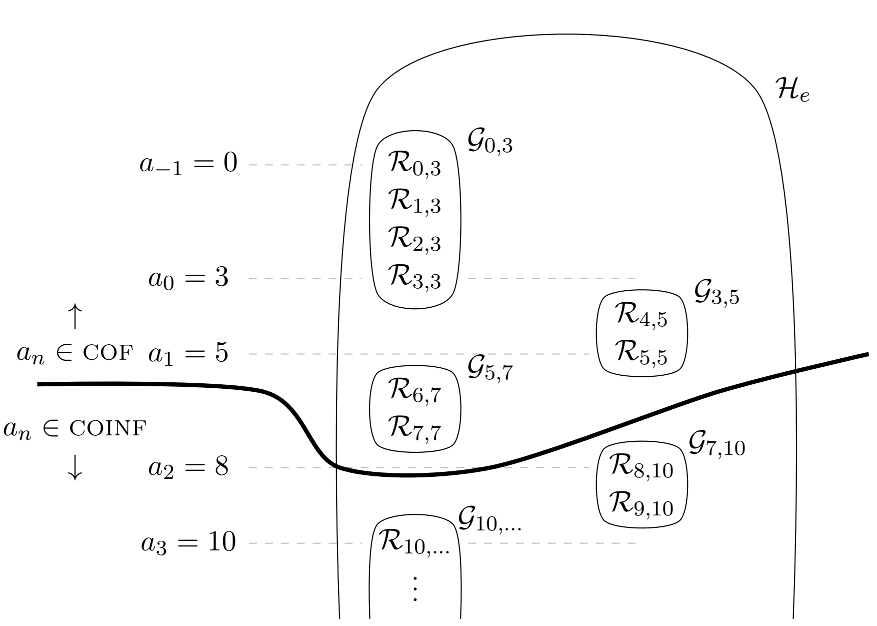


Figure 5: *An example sequence* $(a_n) = (0, 3, 5, 7, 10, ...)$. *The structure of* $\mathcal{H}_e$ *follows by definition from* (a_n). *Note that this figure should be read like a Venn diagram: the* $\mathcal{G}_{j,i}$ *and* $\mathcal{R}_{n,i}$ *are not elements of* $\mathcal{H}_e$, *but subsets. Languages that are in* $\mathcal{R}_{n,i}$ *(i.e. subsets of* F_n*) are the elements of* $\mathcal{H}_e$. *The meaning of the thick line is explained on page 26.*

to be the smallest even number greater than or equal to x, and define $l(x)$ to be the smallest odd number greater than or equal to x. Using these functions and the construction of $\mathbb{R}_{n,i}$, define the family $\mathcal{G}_{x,y}$ with $x, y \in \mathbb{N}$ as follows:

$$\mathcal{G}_{x,y} = \bigcup_{l(x) \leq n \leq u(y)} \mathcal{R}_{n,u(y)}$$

Now, it is finally time to define the family $\mathcal{H}_e$:

$$\mathcal{H}_e := \bigcup_{y \in \mathbb{N}} \mathcal{G}_{a_{y-1},a_y}$$

So, $\mathcal{H}_e$ is the union of a big number of families $\mathcal{R}_{n,a_y}$, which are clustered according to their a_y-values. For an example, see Figure 5.

To recursively find an index for $\mathcal{H}_e$ from the number e, the *Smn*-theorem is used the same way as for finding the indices for the different $\mathcal{R}_{n,i}$-columns. We define a new function h' as

$$h'(e, x) \simeq \begin{cases} 0 & \text{if there exists an } \mathcal{R}_{n,i} \subseteq \mathcal{H}_e \\ & \text{for which } x \text{ is the index of a column} \\ \uparrow & \text{otherwise} \end{cases}$$

$h'(e, x)$ can be computed by inspecting all columns of all the sets $\mathcal{R}_{n,i} \subseteq \mathcal{H}_e$ in a similar fashion as in Figure 2: for increasing m, compute the indices of the first m columns of the first m $\mathcal{R}_{n,i}$-sets in $\mathcal{H}_e$, and compare them to x. If a match is found, output 0. Otherwise, keep searching. Again, by the Smn-theorem, the index of the function $\lambda x.h'(e, x)$ (the domain of which is $\mathcal{H}_e$) can be found primitive recursively in e.

Now that $\mathcal{H}_e$ has been defined and it has been demonstrated how its index can be found recursively in e, the only thing that remains to be done is to verify that $e \in P$ if and only if this index is in EXL.

Verification. In this paragraph, it is verified that $e \in P$ if and only if the constructed family $\mathcal{H}_e$ is TxtEx-learnable.

First, suppose that e is not in P. Then, by clause (2) on page 22, for all y, $f(e, y) \in$ COF. Since $(a_n)_{n\in\mathbb{N}}$ is a subsequence of $(f(e, n))_{n\in\mathbb{N}}$, all values a_y are in COF as well. For every n, $\mathcal{H}_e$ contains a subset of the form $\mathcal{R}_{n,a_y}$ which, as argued, contains the language F_n as an element. Therefore, the family $\mathcal{H}_e$ contains $\mathcal{F}$ as a whole, rendering it non-TxtEx-learnable.

For the other direction, suppose that e is in P. Then, by clause (1) on page 22, for all but finitely many values of y, $f(e, y) \in$ COINF. Examining the definition of Q' even closer, one can see that there exists a y_0 (namely the smallest y such that $f(e, y) \in Q$) such that for $y < y_0$, $y \in$ COF and for $y \leq y_0$, $y \in$ COINF. Based on this y_0, the family $\mathcal{H}_e$ can be divided into two seperate parts as in Figure 5. Let us analyse those two parts:

- For $a_y < y_0$, all families $\mathcal{R}_{n,a_y}$ are finite: they contain a finite amount of (finite) subsets of F_n, in addition to the set F_n itself. Because the union of all these families $\mathcal{R}_{n,a_y}$ is also finite, it can be TxtEx-learned by a machine M_1 that has a list of every language and the (sets of) elements that differentiate them from the other languages in the family.

  Every language in this union contains a number less than $\lfloor \frac{n}{2} \rfloor$.
- For $a_y \geq y_0$, the families $\mathcal{R}_{n,a_y}$ are possibly infinite, but the languages they contain are all finite. Thus, the union of all families $\mathcal{R}_{n,a_y}$ used in the construction with $a_y \geq y_0$ is learnable by a machine M_2 that learns the family of all finite sets (by simply returning an index for the language containing only and all elements it has seen so far).

  No language in this union contains numbers less than $\lfloor \frac{n}{2} \rfloor$.

Now, a *single* Turing machine can identify any language in $\mathcal{H}_e$ by determining whether a number less than $\lfloor \frac{n}{2} \rfloor$ has appeared in the text, and, accordingly, calling M_1 or M_2 on the input. Thus, if $e \in P$, $\mathcal{H}_e$ is TxtEx-learnable.

In this section, we have given a computable way to determine the index of a language family $\mathcal{H}_e$, dependent on e, which is learnable if and only if $e \in P$. So this arbitrary Σ_4-set P is reducible to EXL, the set of indices for families that are TxtEx-learnable, and EXL is Σ_4-hard.

4.2 Finite learning

In finite learning, also called TxtFin-learning, the learner is required not only to converge toward a correct hypothesis, but also to be self-monitoring in the sense that it is able to tell exactly when it has reached a stable state [10]. Before it has, it will only output the symbol ?. As soon at is enters the convergence state, it will return a 'real' hypothesis, which must immediately be correct.

The TxtFin-criterion for identifying a language is stricter than TxtEx and, accordingly, less language families can be learned. The class finl of TxtFin-identifiable language families (again, coded as a subset of $\mathbb{N}$) is simpler in structure than exl: the following subsections provide proof that finl is Σ_3-complete.

4.2.1 Upper bound: finl $\in \Sigma_3$

For the upper bound, a Σ_3 formula is to be formed that exactly describes TxtFin-learnability. Again, let $\mathcal{L}$ be a uniformly recursively enumerable language family $\{L_1, L_2, ...\}$ enumerated by the function ϕ_e. Now e is in finl if and only if there is a Turing machine M_k such that for all languages $L_i \in \mathcal{L}$ the following two statements hold:

1. There exists a finite string, σ_i, that is in some text for L_i, on which M_k returns a real hypothesis (other than ?).
2. On any text for L_i, the first real hypothesis that M_k outputs (if it exists) is correct.

Note that M_k does not necessarily learn $\mathcal{L}$. If M_k is presented with a text for some L_i that does not contain σ_i, it might keep returning ?. However, the existence of M_k implies the existence of *another* Turing machine that *does* learn all languages in $\mathcal{L}$. This machine M, on an input τ, constructs all possible beginnings (maximum length $\text{lh}(\tau)$) of texts for the contents of τ and for any subset of those contents. It then runs M_k on these (parts of) texts. If M_k outputs a (first) real hypothesis on any of these simulated texts, this hypothesis must be correct by the second requirement on M_k, so M can simply copy this hypothesis and be done. If not, M outputs ? and waits for the next input. If M is presented with a text for L_i, it will eventually receive an input from which it will simulate a text starting with σ, and so it will eventually come across a real hypothesis of M_k and output that. So, for every text for L_i, M will output a real hypothesis at some point, and it will be correct.

From the above argument, it follows that if the two statements hold for some M_k and all L_i, then $\mathcal{L}$ is TxtFin-learnable. The other direction is clear immediately: if a language is TxtFin-learnable, the two conditions are met.

Now it is time to transform the textual statements into logical formulae A and B:

1. The fact that a string σ is part of a text for L_i is equivalent to the statement that its contents is a subset of L_i, so the first requirement on M_k is expressed by the formula A, defined as

$$\exists\sigma\Big((\text{content}(\sigma) \subseteq L_i) \wedge M_k(\sigma)\downarrow \wedge (M_k(\sigma) \neq ?)\Big)$$

2. The second requirement states that for strings α, if it is part of a text for L_i and it is the first string of the text for which M_k outputs a real hypothesis, then that hypothesis is correct, as expressed by the formula B:

$$\begin{aligned}\forall\alpha\Big(\quad &\Big((\text{content}(\alpha) \subseteq L_i) \wedge \forall\tau \prec \alpha(M_k(\tau) = ?)\\ &\wedge(M_k(\alpha)\downarrow) \wedge (M_k(\alpha) \neq ?)\Big)\\ &\rightarrow (W_{M_k(\alpha)} = L_i) \quad\Big)\end{aligned}$$

The formula describing FINL,

$$\exists k \forall i (A \wedge B),$$

is Σ_3, as verified in Appendix A. Statements like "content$(\sigma) \subseteq L_i$" will be formalized even further in that verification.

4.2.2 Lower bound: FINL is Σ_3-hard

To provide a lower bound to the complexity of FINL, Beros shows that FINL is at least as hard as any other problem in Σ_3 by reducing an arbitrary Σ_3-set P in Σ_3 to it. Combined with the upper bound result, this proves that FINL is Σ_3-complete.

Fix an arbitrary Σ_3-set P. By definition,

$$P = \{e \mid \exists x \forall y \exists z R(e, x, y, z)\}$$

with R some primitive recursive predicate. Now for an index e, a recursive construction of a family $\mathcal{G}_e = \{G_0, G_1, G_2, ...\}$ will be given such that $e \in P$ if and only if $\mathcal{G}_e$ is TxtFin-learnable. The elements of the sets in $\mathcal{G}_e$ will be read as codes for pairs (see Figure 6), and an element $\langle a, b\rangle$ with $a \geq 0$ is called an *a-label.* Eventually, if there exists an x such that $\forall y \exists z R(e, x, y, z)$, every set G_n will have its own unique x-label for that value of x. A learner can then identify the languages based on those labels.

$$\begin{array}{|c|c|c|c|c}
\langle -1,0\rangle & \langle 0,0\rangle & \langle 1,0\rangle & \langle 2,0\rangle & \cdots \\
\langle -1,1\rangle & \langle 0,1\rangle & \langle 1,1\rangle & \langle 2,1\rangle & \cdots \\
\langle -1,2\rangle & \langle 0,2\rangle & \langle 1,2\rangle & \langle 2,2\rangle & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Figure 6: *Natural numbers ordered according to the pairs they code. The leftmost column is used for the creation of anti-labels.*

$$\boxed{\begin{array}{l}
G_1 = \{\langle 0,0\rangle, \boldsymbol{\langle 0{,}3\rangle}, \langle 5,2\rangle\} \\
G_2 = \{\boldsymbol{\langle 0{,}3\rangle}, \langle 2,7\rangle\} \\
G_3 = \{\langle 0,2\rangle, \langle 2,7\rangle\} \\
G_4 = \{\langle 0,1\rangle, \boldsymbol{\langle 0{,}3\rangle}\}
\end{array}} \Rightarrow \boxed{\begin{array}{l}
G_1 = \{\langle 0,0\rangle, \langle 0,3\rangle, \langle 5,2\rangle\} \\
G_2 = \{\langle 0,3\rangle, \langle 2,7\rangle, \langle \boldsymbol{-1}\ \boldsymbol{,0}\rangle\} \\
G_3 = \{\langle 0,2\rangle, \langle 2,7\rangle\} \\
G_4 = \{\langle 0,1\rangle, \langle 0,3\rangle, \langle \boldsymbol{-1}\ \boldsymbol{,0}\rangle\}
\end{array}}$$

Figure 7: *An example of the anti-labelling of sets with the same x-label. In this example, the set G_1 receives the anti-label $\langle -1, 0\rangle$ for the* 0*-label $\langle 0, 3\rangle$. All* other *sets with that same 0-label (namely G_2 and G_4) receive the element $\langle -1, 0\rangle$. If, later on, other sets are anti-labelled (for example, the set G_2 with respect to the label $\langle 0, 3\rangle$ or $\langle 2, 7\rangle$) the next unused anti-label, in this case $\langle -1, 1\rangle$, is used.*

For any x-label, so-called *anti-labels* can be formed: an anti-label for $\langle x, k\rangle$ is a pair of the form $\langle -1, b\rangle$ that is added to all sets G_n with the label $\langle x, k\rangle$, except for one set $G_{n'}$. $\langle -1, b\rangle$ 'anti-labels' $G_{n'}$, because a learner cannot use $\langle -1, b\rangle$ to identify $G_{n'}$: it cannot wait for it 'not to appear' in a text for the language. For an example of anti-labelling, see Figure 7.

Again, the family $\mathcal{G}_e$ is constructed in stages. As in Section 4.1.2, the entire family $\mathcal{G}_e$ cannot finitely be constructed, but by the *Smn*-theorem, its index can.

To construct $\mathcal{G}_e$ from e, systematically go through all triples $\langle x, y, z\rangle$. At each stage, determine whether the current triple satisfies the following conditions:

Condition 1: $R(e, x, y, z)$ holds.

Condition 2: For all $y' < y$, some triple $\langle x, y', z'\rangle$ has been encountered in a previous stage such that $R(e, x, y', z')$. But, for y itself, the current triple is the first triple $\langle x, y, z'\rangle$ such that $R(e, x, y, z')$.

If these conditions are met, this is a step towards finding suitable z for *all y* – this x may be a good candidate for satisfying $\forall y \exists z R(e, x, y, z)$. Consider the following two cases:

Case 1 Both conditions are met. Three actions are carried out in this case:

1. If two sets G_m and G_n share the same x-label, synchronize those sets by storing all elements of G_n into G_m and vice versa. In effect, both sets now equal $G_m \cup G_n$.
2. All sets with an x-label receive an anti-label for that label. So, the sets that were synchronized in the previous item will differ because of their anti-labels.
3. The first set G_n that does not yet have an x-label receives a new, unique x-label.

Case 2 One or both of the conditions are not met. For each x-label already in use, a new set is created by copying the first G_n with that x-label, and giving the new set an anti-label. Doing so prevents the sets in $\mathcal{G}_e$ from being recognizable by their x-labels.

$\mathcal{G}_e$ is the family that results if this construction is carried out into infinity, thereby examining all triples $\langle x, y, z \rangle$. Now, let us verify that it is indeed learnable if and only if e is in P.

For the first direction, suppose that $e \in P$. This means that, for some x, $\forall y \exists z R(e, x, y, z)$. The actions for Case 1 are then carried out infinitely many times for that x. By action 3, all sets in $\mathcal{G}_e$ will eventually receive an x-label. Even though sets with the same x-label keep being anti-labelled, every such anti-label is synchronized in some subsequent step of the construction. So, in infinity, sets with the same x-label become equal and 'collapse' into a single set. All sets in $\mathcal{G}_e$ can thus be recognized by their x-labels. Moreover, once a machine encounters an x-label, it can immediately be sure of its answer, which is required for TxtFin-learning.

For the other direction, suppose that $e \notin P$. Then for all x, Case 1 will be true only finitely many times. After that, the action of Case 2 will create infinitely many copies of some of the G_n, that differ only by their anti-labels. Such a G_n cannot be TxtFin-learned by any machine M, because for M to be *sure* it is dealing with G_n, it must wait for all elements that anti-label G_n's copies to appear in the text. If some anti-label has not yet been encountered, it may be dealing with one of G_n's copies. Since there are infinitely many of these anti-labels, M cannot give a conclusive answer in a finite amount of time *and* be sure of its answer. So M cannot learn G_n, and hence the entire family $\mathcal{G}_e$ is not TxtFin-learnable.

4.3 Behaviorally correct learning

In behaviorally correct learning or TxtBC-learning, the learner is required to, in the limit, settle on the language it is presented with. Unlike in TxtEx-learning, however, it is allowed to alternate between the different indices for that language. BCL denotes the class of all language families learnable under the TxtBC-criterion. In this subsection, the proof of Σ_5-completeness of BCL is described.

4.3.1 Upper bound: BCL $\in \Sigma_5$

BCL is given an upper bound to its complexity by providing it with a Σ_5 formula. Like in the classification of EXL, this formula will not directly translate the definition of TxtBC-learning into a logical formula. The following observation is used: if a language family is TxtBC-learnable from Δ_2-texts, then it is TxtBC-learnable from arbitrary texts [2].

First, the definition and structure of a Δ_2-text is given. For a text f, define the *graph* of that text to be the set

$$\{\langle x, y\rangle \mid f(x) = y\}$$

If this graph is a Δ_2-set, then f is Δ_2 as well. As shown in [2], such a text f can be assumed to have been defined from the uniformly recursive functions[5] $\{f_0, f_1, f_2, ...\}$:

$$f(x) := \lim_{s\to\infty} f_s(x)$$

Of course, all these limits need to exist for f to become a total function. Each f_s represents a text, and these text converge toward a single, 'final' text f. A value $f_s(x)$ is called *stable* if it equals the final $f(x)$.

The fact that f is a Δ_2 text for a language L_i of a uniformly recursively enumerable family $\mathcal{L} = \{L_0, L_1, L_2, ...\}$ (with index e) is expressed by the conjunction $A = A_1 \wedge A_2 \wedge A_3$ of the following formulas.

- f needs to be defined for all x, i.e. the limits $\lim_{s\to\infty} f_s(x)$ must exist:

$$A_1 := \forall x \exists s \forall s' > s\big(f_{s'}(x) = f_s(x)\big)$$

- All numbers enumerated by f must be in L_i: that is, if some value $f_s(x)$ is stable, that value must be in L_i.

$$A_2 := \forall x \forall s\Big(\forall s' > s\big(f_{s'}(x) = f_s(x)\big) \rightarrow f_s(x) \in L_i\Big)$$

[5]A set of functions $\{f_0, f_1, f_2, ...\}$ is uniformly recursive if there exists a recursive function g such that for all $s \in \mathbb{N}$, $f_s = \lambda x.g(s, x)$

- All numbers in L_i are enumerated by f: that is, they must occur as a *stable* value $f_s(x)$.

$$A_3 := \forall x\Big(x \in L_i \to \exists y \exists s\Big(f_s(y) = x \wedge \forall s' > s\big(f_{s'}(y) = f_s(y)\big)\Big)\Big)$$

The formula A describes what it means for a text f to be a Δ_2-text for a language L_i. Next, let us explicate the conditions for a machine M_k to learn a language L_i from f. There are two conditions to be met:

1. M_k should converge toward a single language (but not toward a single index) on f. That is, for some x, M_k should return indices for that language on all *stable* inputs of length at least x. This is expressed by the following formula:

$$\exists n \forall s \forall n' > n\Big(\forall s' > s\big(f_{s'} \upharpoonright n' = f_s \upharpoonright n'\big) \to (W_{M_k(f_s \upharpoonright n)} = W_{M_k(f_s \upharpoonright n')})\Big)$$

2. The language M_k converges to should be L_i: if, on some input $f_s \upharpoonright n$, M_k returns the same index as it will on all stable inputs of at least length n, then that index should be an index for L_i.

$$\forall n \forall s\Big(\forall n' > n \forall s' > s\Big((f_{s'} \upharpoonright n' \text{ is stable}) \to W_{M_k(f_{s'} \upharpoonright n')} = W_{M_k(f_s \upharpoonright n)}\Big) \\ \to (W_{M_k(f_s \upharpoonright n)} = L_i) \Big)$$

Here, the statement "$f_{s'} \upharpoonright n'$ is stable" can be translated as

$$\forall s'' > s'(f_{s''} \upharpoonright n' = f_{s'} \upharpoonright n')$$

The resulting formula that describes TxtBC-learnability,

$$\exists k \forall g, i\Big((g \text{ defines a text } f \text{ for } L_i) \to \\ \Big((M_k \text{ converges on } f) \wedge (\text{if } M_k \text{ converges on} f, \text{ it is toward } L_i)\Big)\Big),$$

is Σ_5, as shown in Appendix A. Consequently, BCL is in Σ_5.

4.3.2 Lower bound: BCL is Σ_5-hard

In order to prove that BCL is Σ_5-hard, an arbitrary Σ_5-set P should be reduced to it. This reduction proof consists of three parts: first, as in the hardness proof of EXL, we bring structure to the set P using the fact that COINF is Π_3-complete. Then, a family $\mathcal{F}_{a,b}$ is constructed, the learnability of which will depend on whether or not $b \in$ COINF. Finally, using the families $\mathcal{F}_{a,b}$, a family $\mathcal{G}_e$ will be constructed from e such that $e \in P$ if and only if $\mathcal{G}_e$ is TxtBC-learnable.

The structure of P. In this paragraph, the structure of an arbitrary Σ_5-set P will be examined by 'unwrapping' it according to its definition in the arithmetical hierarchy. If P is Σ_5, then its complement, $\overline{P}$, is Π_5 and therefore of the form

$$\overline{P} = \{e \mid \forall x((e,x) \in Q)\}$$

for some $Q \in \Sigma_4$. Analogously to Section 4.1.2, define another Σ_4-set Q' from Q:

$$Q' := \{(e,x) \mid \forall x' \leq x((e,x) \in Q)\}$$

Now, it follows that

$$\begin{array}{rcl} e \in P & \Rightarrow & \exists x((e,x) \notin Q) \\ & \Rightarrow & \exists x \forall x' \geq x((e,x) \notin Q') \\ e \notin P & \Rightarrow & \forall x((e,x) \in Q') \end{array}$$

In order to dissect the structure of P even further, observe that Q', being Σ_4, is of the form

$$Q' = \{(e,x) \mid \exists y((e,x,y) \in R)\}$$

for some $R \in \Pi_3$. From R, another Π_3-set R' can again be defined:

$$R' = \{(e,x,y) \mid (e,x,y) \in R \wedge \forall y' < y((e,x,y) \notin R)\}$$

In words, R' contains, for every e and x, only the *smallest* y such that $(e,x,y) \in R$ (if such y exists). Furthermore, from the definition of Q' and R' it follows that

$$\begin{array}{rcl} (e,x) \in Q' & \Rightarrow & \exists! y((e,x,y) \in R') \\ (e,x) \notin Q' & \Rightarrow & \forall y((e,x,y) \notin R') \end{array}$$

Finally, because R' is Π_3, it can be m-reduced to the Π_3-complete set $\textsc{coinf}$. That is, there exists a total recursive function f such that

$$(e,x,y) \in R' \Leftrightarrow f(e,x,y) \in \textsc{coinf}$$

Combining all observations made above, it follows that

$$\begin{array}{rcl} e \in P & \Rightarrow & \exists x \forall x' \geq x \forall y(f(e,x',y) \notin \textsc{coinf}) \\ e \notin P & \Rightarrow & \forall x \exists! y(f(e,x,y) \in \textsc{coinf}) \end{array}$$

Moreover, note that in the first case (if $e \in P$), for all $x' < x$, either for all y, $f(e,x',y) \notin \textsc{coinf}$, or there is exactly one y for which $f(e,x',y) \in \textsc{coinf}$. Hence, in total, for each $e \in P$ there exist only finitely many pairs (x,y) such that $f(e,x,y) \in \textsc{coinf}$.

These observations will be used in the third part of this subsection.

The construction of the subfamilies $\mathcal{F}_{a,b}$. The second part of the proof consists of the construction of language families $\mathcal{F}_{a,b}$, which will eventually become subfamilies of $\mathcal{G}_e$. They will have the following properties:

1. If $b \in$ COINF, then $\mathcal{F}_{a,b}$ is TXTBC-learnable. However, the machine M_a does *not* learn it.

2. The language family that is the union of all families $\mathcal{F}_{a,b}$ with $b \in$ COF is TXTBC-learnable.

Before defining these families, the notion of *speculation* is introduced.

Definition 5. *If a learner, on some input σ, hypothesizes a language that contains elements that are not listed in σ, the learner* speculates *on σ. The elements that are in the hypothesized language but not in σ are called* speculations.

Speculating can be thought of as 'thinking outside the box'. If a learner hypothesizes an infinite language, it is necessarily speculating.

Now, let us move on to the construction of a family $\mathcal{F}_{a,b}$. It will consist of the languages A, B_0, B_1, B_2, et cetera. Depending on the behaviour of M_a, there will be either a finite or an infinite amount of languages B_i. All of these B_i (except for possibly the last) will be labelled with the element $\langle 0, i\rangle$. Additionally, all languages in $\mathcal{F}_{a,b}$ contain the elements $\langle 1, \langle 0, a\rangle\rangle$ and $\langle 1, \langle 1, b\rangle\rangle$ as labels. These labels will make the languages easier to recognize by a learner. No other elements will be of the form $\langle 0, n\rangle$ or $\langle 1, n\rangle$, in order to avoid confusing the learner.

The construction of $\mathcal{F}_{a,b}$ consists of several stages, during which a possibly infinite sequence σ is built up that will serve as (the start of) a text for A. The goal is to design σ in such a way that it is impossible for M_a to identify A from it. Let σ_i denote the part of σ constructed up to the i^{th} stage. At each stage i, an extension $\sigma_i \star \alpha_i$ for σ_i is sought on which M_a speculates. The least speculation of M_a on this extension is called c_i. Now, define D as follows:

$$D := \mathbb{N} \setminus \{c_j \mid j \notin W_b\}$$

where c_j refers to the speculation on $\sigma_j \star \alpha_j$. Every extension α_i may consist only of elements of D.

Eventually, all sets in $\mathcal{F}_{a,b}$ will be finite subsets of D (plus labels), except for at most one of them, which will equal D.

The search for α_i is divided into finite steps (at the nth step, the length and contents of the extensions to be tested are restricted not to exceed n). If it takes n steps to find a new extension, the first n elements of D are stored in B_i. If no extension is ever found, and thus infinitely many steps are carried out, B_i will contain all elements of D.

Once α_i *is* found, several actions are carried out. First, all elements of D that do not exceed the maximum element of α_i are stored in A. Then,

σ_{i+1} is updated to $\sigma_i \star \alpha_i \star \beta_i$, where β_i is a list of all numbers that were just stored in A. Note that this way, σ eventually contains only and all elements of A. Finally, the construction of B_i ends: it receives its label $\langle 0, i\rangle$ and the construction moves on to the set B_{i+1}.

The construction of $\mathcal{F}_{a,b}$ will have one of two outcomes: if infinitely often a new extension α_i is found, then an infinite number of languages B_i are constructed. They will all eventually finish and be labelled. If, at some point, no new extension α_i is ever found, there will only be a finite number of languages B_i. The last of those will be infinite and equal to D. The set A, however, will certainly be finite.

Next, we prove that $\mathcal{F}_{a,b}$ has the desired properties stated at the beginning of this paragraph. These two possible outcomes always have to be considered: either $\mathcal{F}_{a,b}$ contains infinitely many languages, or finitely many.

First, suppose that $b \in$ COINF. Then $\mathcal{F}_{a,b}$ should be TxtEx-learnable, but *not* by the machine M_a.

Case 1 Suppose $\mathcal{F}_{a,b}$ is infinite. In this case, learning is simple, because all B_i-sets are labelled. A learner can hypothesize A until it encounters such a label.

However, M_a fails to learn $\mathcal{F}_{a,b}$, because it does not identify A from the infinite sequence σ. Infinitely often, M_a speculates on σ. However, because W_b is coinfinite, infinitely many of these speculations will not be part of A. So infinitely often, M_a will hypothesize a language that does not equal A.

Case 2 Suppose $\mathcal{F}_{a,b}$ is finite. Any finite family can be learned by a machine that has a list of those (sets of) elements that make each language in the family unique. Hence, $\mathcal{F}_{a,b}$ is learnable.

This time, M_a fails to learn $\mathcal{F}_{a,b}$ because it cannot identify the infinite language B_k from any text f that starts with the (finite) sequence σ. Because M_a never speculates on any extension of σ, it will, in the limit, only hypothesize finite languages on f. However, B_k is infinite, so M_a does not identify B_k from f.

For the second requirement on $\mathcal{F}_{a,b}$, a *single* machine N has to be constructed that learns all languages in the union of those $\mathcal{F}_{a,b}$ with $b \in$ COF. So this time, we cannot construct two different machines for the two different cases. N has to be flexible enough to deal with both cases.

N will first wait for the labels $\langle 1, \langle 0, a\rangle\rangle$ and $\langle 1, \langle 1, b\rangle\rangle$ to give away which subfamily it is learning from. Then, it will imitate the construction of the family $\mathcal{F}_{a,b}$, one stage at a time. If the text presents a label $\langle 0, i\rangle$, N will of course hypothesize the B_i it has constructed so far. Otherwise it will have to guess whether the text it is presented with is a text for A or for the (possibly non-existent) last set B_k. N will always view the set B_i it is currently constructing as a candidate for B_k. In order to choose between A and B_k, N will test whether the elements it has seen so far are all contained in the part of A it has constructed up to this point. If so, it hypothesizes A. If not, it hypothesizes $A \cup B_k \backslash \langle 0, k\rangle$, where B_k is the language that N at that moment views as the 'last'.

Clearly, N is able to recognize which family $\mathcal{F}_{a,b}$ the language it is learning belongs to. To see why N will succeed in learning the language itself, again consider the two possible forms of this $\mathcal{F}_{a,b}$. The universal strategy of N will work in both cases.

Case 1 Suppose $\mathcal{F}_{a,b}$ is infinite. In this case, all B_i-languages will be correctly identified by N, because they are all labelled. If N is presented with A, again consider two possibilities. If A is finite, N will correctly identify it because, in the limit, it will have constructed all of A and so the test of whether or not the elements it has been presented with match the element of A will eventually always come up positive. If A is infinite, then it will equal the set D. Every $B_k \backslash \{\langle 0, k\rangle\}$ will be a (finite) subset of D, so $A \cup B_k \backslash \{\langle 0, k\rangle\}$ equals A. So in this case, it does not matter whether N hypothesizes A or $A \cup B_k \backslash \{\langle 0, k\rangle\}$. Even alternating infinitely many times between these two hypotheses is allowed in TxtBC-learning.

Case 2 Suppose $\mathcal{F}_{a,b}$ is finite. In this case, there will be a 'last' set B_k which will equal D and is not yet labelled (again, all labelled sets B_i are easily identified). A will be a finite set and thus, in the limit, N will have constructed all of A. If presented with a text for A, it will eventually never come across elements that are not in its construction for A. If N is presented with a text for the infinite set B_k, it will. In that case, N will output $A \cup B_k \backslash \{\langle 0, k\rangle\}$. Since A is a subset of D and B_k is not yet labelled, $B_k = A \cup B_k \backslash \{\langle 0, k\rangle\}$, so this hypothesis is also correct. Therefore, N is able to learn all languages in $\mathcal{F}_{a,b}$.

The construction of the family $\mathcal{G}_e$. Now, the first and second parts of this proof are used in the construction of $\mathcal{G}_e$, a family that will be TxtBC-learnable if and only if $e \in P$. Using the function f from the first part of the proof, define

$$\mathcal{G}_e := \bigcup_{x,y \in \mathbb{N}} \mathcal{F}_{x,f(e,x,y)}$$

First, suppose that $e \in P$. As argued, there exist only finitely many pairs (x_i, y_i) such that $f(e, x_i, y_i) \in \text{COINF}$. The families $\mathcal{F}_{x,f(e,x,y)}$ are TxtBC-learnable, say by a machine M_{a_i}. The union of all other families in $\mathcal{G}_e$ are learnable by the machine N constructed earlier. $\mathcal{G}_e$ is now learnable as follows: on an input σ, call N, unless the labels $\langle 1, \langle 0, x_i\rangle\rangle$ and $\langle 1, \langle 1, f(e, x_i, y_i)\rangle\rangle$ have been encountered for one of the pairs (x_i, y_i), in which case the machine M_{a_i} is called. Since there are only finitely many of those pairs, these instructions can be hard wired into the Turing machine.

Next, suppose that $e \notin P$, and suppose that some machine M_a learns the family $\mathcal{G}_e$. Because e is not in P, there exists a y such that $f(e, a, y) \in$ COINF. By definition of the family $\mathcal{F}_{a,f(e,a,y)}$, it cannot be learned by M_a. Since it is part of the family $\mathcal{G}_e$, the entire family $\mathcal{G}_e$ cannot be learned by M_a, a contradiction. Consequently, $\mathcal{G}_e$ is not TxtBC-learnable.

4.4 Anomalous learning

In anomalous learning, or TxtEx*-learing, the learner is again required to settle on a single index. The set this index represents, however, is allowed to differ slightly from the language that is presented. Specifically, the symmetric difference between the two sets should be finite.

The class EXL* of indices for families learnable under this criterion is the fourth and final class that Beros positions into the arithmetical hierarchy. It is, like BCL, exactly in Σ_5.

4.4.1 Upper bound: EXL* $\in \Sigma_5$

TxtEx* closely resembles TxtEx, so the formula that describes learnability in this model is very similar. The formula is again of the form

$$\exists k \forall i \forall a \Big((M_k \text{ total}) \wedge \Big(((\phi_a \text{ total}) \wedge (\phi_a \text{ enumerates } L_i)) \rightarrow (A \wedge B) \Big) \Big)$$

only now the formula B is slightly different. The formula A still represents the fact that M_k should eventually converge on ϕ_a. The second formula, B, again represents the fact that at each point in time, if M_k has converged, its outputs should be correct. Only for this learning criterion, 'correct' has a different meaning: it means that the finite symmetric difference of the hypothesized language and L_i, denoted $W_{M_k(\phi_a \restriction n)} \triangle L_i$, is finite.

The indices of all finite sets can be enumerated in a computable manner. This can be done in stages: at each stage n, list all sets with at most n elements and no elements greater than n. This way, all sets will be listed, albeit with repetition. Let h be the recursive function that enumerates these indices – so $\{W_{h(0)}, W_{h(1)}, ...\}$ is the set of all finite sets. Now, the formula B reads as follows:

$$\exists l \forall n \Big(\forall m \Big(m > n \rightarrow M_k(\phi_a \restriction m) = M_k(\phi_a \restriction n) \Big) \rightarrow \Big(W_{M_k(\phi_a \restriction n)} \triangle L_i = W_{h(l)} \Big) \Big)$$

The expression $W_{M_k(\phi_a \restriction n)} \triangle L_i = W_{h(l)}$ is Π_2, as is shown in Appendix A. In this Appendix the conversion to the prenex normal form of the describing formula as a whole is also carried out. The describing formula is Σ_5, as expected.

4.4.2 Lower bound: EXL* is Σ_5-hard

The Σ_5-hardness proof of EXL* is very similar to that of BCL in Section 4.3.2. In fact, the first part (bringing structure to an arbitrary Σ_5-set P) and the third part (construction of $\mathcal{G}_e$ from $\mathcal{F}_{a,b}$) of the proof are exactly equal. So, we aim to again construct families $\mathcal{F}_{a,b}$ with the following properties:

1. If $b \in$ COINF, then $\mathcal{F}_{a,b}$ is TXTEX*-learnable. However, the machine M_a does *not* learn it.

2. The language family that is the union of all families $\mathcal{F}_{a,b}$ with $b \in$ COF is TXTEX*-learnable.

Given such families, the construction of a family $\mathcal{G}_e$ such that $e \in P$ if and only if $\mathcal{G}_e$ is TXTEX*-learnable is identical to the construction in Section 4.3.2.

So let us now construct the family $\mathcal{F}_{a,b}$ with the aforementioned mentioned properties. $\mathcal{F}_{a,b}$ will resemble the structure of the family in Section 4.3.2. It will consist of the languages $A, L_0, R_0, L_1, R_1, L_2, R_2$, et cetera, possibly infinitely many. All these languages will again be labelled to show which family they belong to. In addition to this, all languages, except for possibly the last pair L_n and R_n receive their own labels. The rest of the construction occurs outside of these labels, which are all of the form $\langle 0, x\rangle$ or $\langle 1, x\rangle$.

Eventually, each L_i will consist mostly of even numbers and each R_i mostly of odd numbers. Depending on the contents of W_b, the symmetric difference between the last L_n and R_n, if they exist, is either finite or infinite. If their symmetric difference is finite, there is no need for a machine to discern between them in order to be able to learn them under the TXTEX*-criterion.

Again, the construction of $\mathcal{F}_{a,b}$ is carried out in stages. During these stages, a string σ will be built up that will serve as a text for A from which M_a fails to identify A. σ_i will denote this sequence as constructed up to stage i. At each stage i, an extension $\sigma_i \star \alpha_i$ for σ_i is sought, but unlike in the hardness proof of BCL, this extension should be such that $M_a(\sigma_i \star \alpha_i) \neq M_a(\sigma_i)$, i.e., M_a changes its hypothesis. Once such an extension has been found, all numbers less than or equal to the maximum element of α_i are stored in A. Then, σ_{i+1} is set to be $\sigma_i \star \alpha_i \star \beta_i$, where β_i is a list of all numbers that were just stored in A. This way, σ will eventually become a text for A. Finally, the construction of the sets L_i and R_i is ended by labelling them, and the construction of the next pair, L_{i+1} and R_{i+1} is initiated by storing all elements of σ_{i+1} in both sets.

The search for a suitable α_i, like in Section 4.3.2, is a possibly infinite one and is therefore divided into finite steps. At each step j, the least even number k is found such that both k and $k+1$ have not yet been used in the construction of $\mathcal{F}_{a,b}$. If $j \in W_b$, both k and $k+1$ are stored in the sets L_i and R_i. Otherwise, k is stored in L_i, while $k+1$ is stored in R_i.

The construction of $\mathcal{F}_{a,b}$ will either result in a family with finitely many languages, or one with infinitely many. In the last case they will all be labelled. Now, it is time to verify that $\mathcal{F}_{a,b}$ meets the conditions stated on page 38. We will always distinguish between the possibilities of a finite and an infinite family.

First, suppose that $b \in \textsc{coinf}$. Then $\mathcal{F}_{a,b}$ should be TxtEx*-learnable, but not by the machine M_a. Two cases are treated separately:

Case 1 Suppose an extension for σ_i is found infinitely often. Then, all sets are labelled, so a machine can easily TxtEx*-learn $\mathcal{F}_{a,b}$.

However, M_a does not learn $\mathcal{F}_{a,b}$: σ represents a text for A, but is constructed in such a way that M_a changes its hypothesis infinitely often. So, M_a does not converge toward a single hypothesis on this text for A and hence fails to identify A.

Case 2 Suppose for some σ_i, no new extension is found. Then $\mathcal{F}_{a,b}$ contains only finitely many languages and is surely learnable by a machine that has a list of which (finite sets of) elements to identify each of those languages from.

Again, however, M_a does not learn $\mathcal{F}_{a,b}$: σ is a locking sequence (see page 21) for M_a and any language that contains all elements of σ. The last sets L_n and R_n are such languages, and hence M_a is unable to discern between them. Because $b \in \textsc{coinf}$, the symmetric difference between L_n and R_n is infinite (there are infinitely many even numbers in $L_n \backslash R_n$, and vice versa, infinitely many odd numbers in $R_n \backslash L_n$). So M_a fails to identify at least one of them from any text starting with σ.

Next, consider the family that is the union of all $\mathcal{F}_{a,b}$ with $b \in \textsc{cof}$. A machine N can learn this entire family by first waiting for the labels to reveal which family $\mathcal{F}_{a,b}$ the language belongs to. Subsequently, the machine waits for labels to give away the specific sets:

- Labelled finite languages L_n and R_n, which all have a finite symmetric difference from the empty set.
- Labelled set A, the index of which can be recursively constructed from a and b.
- If no label is encountered, the language may be one of the unlabelled, infinite, 'last' sets L_n and R_n. Because $b \in \textsc{cof}$, only finitely often has k or $k+1$ been absent from the sets L_n and R_n. Therefore, both sets are cofinite (except for the numbers of the form $\langle 0, x\rangle$ and $\langle 1, x\rangle$ which are used for labelling), so their symmetric difference from the set $\mathbb{N} \backslash \{\langle x, y\rangle \mid x \leq 1\}$ is finite.

In conclusion, N can simply return the correct (that is, correct in terms of TxtEx*-learning) indices based on the labels, and default to $\mathbb{N} \backslash \{\langle x, y\rangle \mid x \leq 1\}$ if no label has (yet) been encountered.

5 Conclusion

The aim of this work has been twofold. Firstly, to provide an overview of Gold's learning theory and investigate its relation to human language learning, and secondly, to examine four identification criteria for learning and the complexity (in terms of the arithmetical hierarchy) of the classes of languages families learnable under those criteria.

In the analysis of Gold's model for learning, it became clear that it is a very idealized form of learning that differs from human language acquisition in some fundamental ways. In order to adapt the model so that it more closely resembles human language acquisition, more information about the way humans learn a language is required. Although the field of linguistics is constantly progressing, current knowledge is inadequate for the precise mathematical modelling required in learning theory.

Three variations on Gold's model were considered: TxtFin-learning, which is a more limited form of learning, TxtBC-learning, an extensional form of learning, and TxtEx*-learning, where the learner is allowed to make small mistakes. Following Beros [2], the following results regarding the complexity of the learnable language family classes have been found:

- EXL, the class of language families learnable under the (original) TxtEx-criterion, is m-complete in Σ_4.
- FINL, the class of language families learnable under the TxtFin-criterion, is m-complete in Σ_3.
- BCL, the class of language families learnable under the TxtBC-criterion, is m-complete in Σ_5.
- EXL*, the class of language families learnable under the TxtEx*-criterion, is m-complete in Σ_5.

These facts were all proven by providing an upper bound (a describing formula) and a lower bound (reducing an arbitrary Σ_3-, Σ_4- or Σ_5-set) to the class, thereby establishing the exact location in the arithmetical hierarchy.

While these results support the intuition that more sophisticated forms of learning fall into a higher category of the arithmetical hierarchy, this intuition is insufficiently grounded in [2]. For example, the class of all (uniformly recursively enumerable) language families would take a very sophisticated machine to learn, but is fairly simple in structure: it is simply the set of all indices for total functions, because each such function describes a language family. One could argue that under no reasonable learning criterion, all language families would be learnable, and that this example is therefore irrelevant. However, it demonstrates that the correlation between the complexity of the learning process and the complexity of the class of learnable language families is not as self-evident as Beros suggests. Additional evidence for this correlation is needed.

Suppose this classification *is* a good measure of complexity for the learning process. One can then expect the complexity of the class of language families learnable by humans to be at an even higher level than the relatively simple classes BCL and EXL*, because human language acquisition is subject to much more subtle criteria than the learning criteria discussed in this work.

The fact that for these simple learning criteria, the question of whether a certain language family is learnable is already well beyond decidable, does not bode well for the case of human language acquisition, even if we would ever be able to adequately model it. It suggests that learning theory may not have any practical applications in the near future in terms of learning machines. This, of course, does not mean that it cannot be of any theoretical interest to the field of artificial intelligence.

References

[1] L. Becerra-Bonache and T. Yokomori. Learning mild context-sensitiveness: Toward understanding children's language learning. In *Grammatical Inference: Algorithms and Applications*, pages 53–64. Springer, 2004.

[2] A.A. Beros. Learning theory in the arithmetic hierarchy. arXiv:1302.7069 [math.LO], 2013.

[3] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, 1975.

[4] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25(2):193–220, 1983.

[5] R.G. Downey and D.R. Hirschfeldt. *Algorithmic randomness and complexity.* Springer, 2008.

[6] E. M. Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[7] S.C. Kleene. *Introduction to Metamathematics.* North Holland Publishing Co., 1952.

[8] G. F. Marcus. Negative evidence in language acquisition. *Cognition*, 46 (1):53–85, 1993.

[9] J.C. Martin. *Introduction to Languages and The Theory of Computation.* McGraw-Hill, fourth edition, 2010.

[10] D. N. Osherson, M. Stob, and S. Weinstein. *Systems that learn: an introduction to learning theory for cognitive and computer scientists.* Learning, Development and Conceptual Change. MIT press, 1986.

[11] G.K. Pullum and B.C. Scholz. Recursion and the infinitude claim. *Recursion in human language*, (104):113–138, 2010.

[12] M. Sipser. *Introduction to the theory of computation.* Thomson Course Technology, second edition, 2006.

[13] Soare. *Recursively enumerable sets and degrees: A study of computable functions and computably generated sets.* Perspectives in Mathematical Logic. Springer-Verlag, 1987.

[14] J. van Oosten. Basic computability theory. 1993, revised 2013.

Appendices

A Prenex normal forms of formulas

In this appendix, some fomulas constructed throughout the text will be treated in more detail. Specifically, they will be converted to prenex normal form in order to explicitly show which level of the hierarchy they belong to. In doing so, the following equivalences will be used:

$$(\forall x A) \wedge B \equiv \forall x(A \wedge B) \tag{1}$$
$$(\exists x A) \wedge B \equiv \exists x(A \wedge B) \tag{2}$$
$$(\forall x A) \vee B \equiv \forall x(A \vee B) \tag{3}$$
$$(\exists x A) \vee B \equiv \exists x(A \vee B) \tag{4}$$
$$(\forall x A) \rightarrow B \equiv \exists x(A \rightarrow B) \tag{5}$$
$$(\exists x A) \rightarrow B \equiv \forall x(A \rightarrow B) \tag{6}$$
$$A \rightarrow \forall x B \equiv \forall x(A \rightarrow B) \tag{7}$$
$$A \rightarrow \exists x B \equiv \exists x(A \rightarrow B) \tag{8}$$
$$\neg \forall x A \equiv \exists x \neg A \tag{9}$$
$$\neg \exists x A \equiv \forall x \neg A \tag{10}$$

For these equivalences to be used correctly, no unbound variable may become bound after performing an equivalence operation. For example, equivalence (1) may only be used if the variable x does not occur freely in B.

Because most of the formulas will be tediously long and their meaning is not of interest for determining their hierarchy level, a notational system will be used that allows for shorter formulas. In this system a box ($\Box$) will indicate any recursive predicate. In order to make clear which quantifiers are moved in which step of the process, the quantifiers will be numbered *before* they are moved, and both highlighted and numbered *after*. Subsequent quantifiers of the same type will be merged into one. Note that bounded quantifiers do not have to be taken into account, because for all n, Σ_n and Π_n are closed under bounded quantification [13].

For example, the formula

$$\forall x(\exists y(y < x) \rightarrow x \neq 0)$$

is brought to prenex normal form in the following scheme:

$$\begin{aligned} & \forall(\exists_1 \Box \rightarrow \Box) \\ \equiv\ & \forall\, \boxed{\forall_1}\, \Box \\ \equiv\ & \forall \Box \end{aligned}$$

and it follows that this formula is Π_1. Note how in the first step, after moving the $\exists_1$ outside of the brackets, the matrix is $\square \to \square$. However, since this is recursive as a whole, it is immediately replaced by a single square.

A.1 The description of EXL is Σ_4

The full formula for EXL is built up from several subformulas that are listed on pages 18 and 19. They will be converted to prenex normal form one by one.

$$\begin{aligned} & \forall x \exists y T(1,k,x,y) && (1) \\ \equiv\ & \forall\exists\square \end{aligned}$$

$$\begin{aligned} & \forall x \exists y T(1,a,x,y) && (2) \\ \equiv\ & \forall\exists\square \end{aligned}$$

$$\begin{aligned} & \forall x \Big(\exists y \exists z \big(T(1,a,y,z) \wedge U(z)=x\big) \leftrightarrow \exists w T(1,\phi_e(i),x,w)\Big) && (3) \\ \equiv\ & \forall\Big((\exists\exists\square \to \exists\square) \wedge (\exists\square \to \exists\exists\square)\Big) \\ \equiv\ & \forall\Big((\exists_1\square \to \exists\square) \wedge (\exists_2\square \to \exists\square)\Big) \\ \equiv\ & \forall\Big(\boxed{\forall_1}(\square \to \exists\square) \wedge \boxed{\forall_2}(\square \to \exists\square)\Big) \\ \equiv\ & \forall\boxed{\forall_1\forall_2}\Big((\square \to \exists_3\square) \wedge (\square \to \exists_4\square)\Big) \\ \equiv\ & \forall\Big(\boxed{\exists_3}\square \wedge \boxed{\exists_4}\square\Big) \\ \equiv\ & \forall\boxed{\exists_3\exists_4}\square \\ \equiv\ & \forall\exists\square \end{aligned}$$

$$\begin{aligned} & \forall n \Big(\forall m \Big(m > n \to M_k(\phi_a \restriction m) = M_k(\phi_a \restriction n)\Big) \to W_{M_k(\phi_a \restriction n)} = L_i \Big) && (4) \\ \equiv\ & \forall(\forall\square \to \forall_1\exists\square) \\ \equiv\ & \forall\boxed{\forall_1}(\forall_2\square \to \exists_3\square) \\ \equiv\ & \forall\boxed{\exists_2\exists_3}\square \\ \equiv\ & \forall\exists\square \end{aligned}$$

$$\begin{aligned} & \exists s \forall t \Big(t > s \to \Big(M_k(\phi_a \restriction t) = M_k(\phi_a \restriction s)\Big)\Big) && (5) \\ \equiv\ & \exists\forall\square \end{aligned}$$

In (4), the statement $W_{M_k(\phi_a\restriction n)} = L_i$ is replaced with a formula that is almost identical to (3).
The formula $P(e)$ as a whole is now of the form

$$\begin{aligned}
&\exists\forall\Big(\forall_1\exists\Box \wedge \forall_2\Big(\big(\forall\exists\Box \wedge \forall\exists\Box\big) \rightarrow \big(\forall\exists\Box \wedge \exists\forall\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\,\colorbox{lightgray}{$\forall_1\forall_2$}\,\Big(\exists\Box \wedge \Big(\big(\forall_3\exists\Box \wedge \forall_4\exists\Box\big) \rightarrow \big(\forall_5\exists\Box \wedge \exists\forall\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\Big(\exists\Box \wedge \Big(\colorbox{lightgray}{$\forall_3\forall_4$}\,\big(\exists\Box \wedge \exists\Box\big) \rightarrow \colorbox{lightgray}{$\forall_5$}\,\big(\exists_1\Box \wedge \exists_2\forall\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\Big(\exists\Box \wedge \colorbox{lightgray}{$\forall_5$}\Big(\forall_3\big(\exists\Box \wedge \exists\Box\big) \rightarrow \colorbox{lightgray}{$\exists_1\exists_2$}\,\big(\Box \wedge \forall\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\,\colorbox{lightgray}{$\forall_5$}\,\Big(\exists\Box \wedge \colorbox{lightgray}{$\exists_1\exists_2\exists_3$}\Big(\big(\exists_4\Box \wedge \exists_6\Box\big) \rightarrow \big(\Box \wedge \forall\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\Big(\exists_1\Box \wedge \exists_2\Big(\colorbox{lightgray}{$\exists_4\exists_6$}\,\Box \rightarrow \big(\Box \wedge \forall\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\,\colorbox{lightgray}{$\exists_1\exists_2$}\,\Big(\Box \wedge \Big(\exists\Box \rightarrow \big(\Box \wedge \forall_3\Box\big)\Big)\Big)\\
\equiv\ &\exists\forall\exists\Big(\Box \wedge \Big(\exists_1\Box \rightarrow \colorbox{lightgray}{$\forall_3$}\,\Box\Big)\Big)\\
\equiv\ &\exists\forall\exists\Big(\Box \wedge \colorbox{lightgray}{$\forall_1\forall_3$}\,\Box\Big)\\
\equiv\ &\exists\forall\exists\,\colorbox{lightgray}{$\forall_1\forall_3$}\,\Box\\
\equiv\ &\exists\forall\exists\forall\Box
\end{aligned}$$

So, $P(e)$ is indeed Σ_4.

A.2 The description of FINL is Σ_3

First, some subformulas are treated in more detail.

$$\begin{aligned}
&\text{content}(\sigma) \subseteq L_i && (6)\\
\equiv\;& \forall j < \text{lh}(\sigma) \exists y T(1, \phi_e(i), (\sigma)_j, y)\\
\equiv\;& \exists \Box
\end{aligned}$$

– –

$$\begin{aligned}
&M_k(\sigma)\downarrow && (7)\\
\equiv\;& \exists y T(1, k, \sigma, y)\\
\equiv\;& \exists \Box
\end{aligned}$$

– –

$$\begin{aligned}
&W_{M_k(\alpha)} = L_i && (8)\\
\equiv\;& \forall x \big(\exists y T(1, M_k(\alpha), x, y) \leftrightarrow \exists z T(1, \phi_e(i), x, z) \big)\\
\equiv\;& \forall (\exists \Box \leftrightarrow \exists \Box)
\end{aligned}$$

Note that the bounded quantifier in (7) can be ignored, because both Σ_n and Π_n are closed under bounded quantification [13].

Now, the formulas A and B are treated separately:

$$\begin{aligned}
&\exists(\exists_1 \Box \wedge \exists_2 \Box \wedge \Box) && (A)\\
\equiv\;& \exists\, \exists_1 \exists_2\, \Box\\
\equiv\;& \exists \Box
\end{aligned}$$

– –

$$\begin{aligned}
&\forall \big((\exists_1 \Box \wedge \Box \wedge \exists_2 \Box \wedge \Box) \rightarrow \forall (\exists \Box \leftrightarrow \exists \Box) \big) && (B)\\
\equiv\;& \forall \big(\, \exists_1 \exists_2\, \Box \rightarrow \forall \big((\exists_3 \Box \rightarrow \exists \Box) \wedge (\exists_4 \Box \rightarrow \exists \Box) \big) \big)\\
\equiv\;& \forall \big(\exists \Box \rightarrow \forall \big(\, \forall_3\, (\Box \rightarrow \exists \Box) \wedge\, \forall_4\, (\Box \rightarrow \exists \Box) \big) \big)\\
\equiv\;& \forall \big(\exists \Box \rightarrow \forall\, \forall_3 \forall_4\, \big((\Box \rightarrow \exists_1 \Box) \wedge (\Box \rightarrow \exists_2 \Box) \big) \big)\\
\equiv\;& \forall \big(\exists \Box \rightarrow \forall (\, \exists_1\, \Box \wedge\, \exists_2\, \Box) \big)\\
\equiv\;& \forall \big(\exists \Box \rightarrow \forall_3\, \exists_1 \exists_2\, \Box)\\
\equiv\;& \forall\, \forall_3\, \big(\exists_4 \Box \rightarrow \exists \Box \big)\\
\equiv\;& \forall\, \forall_4\, \big(\Box \rightarrow \exists_1 \Box \big)\\
\equiv\;& \forall \exists_1 \Box\\
\equiv\;& \forall \exists \Box
\end{aligned}$$

Finally, the formula that describes FINL is Σ_3:

$$\begin{aligned}
&\exists\forall(A \wedge B)\\
\equiv\ &\exists\forall(\exists\square \wedge \forall_1\exists\square)\\
\equiv\ &\exists\forall\,\forall_1\,(\exists_2\square \wedge \exists_3\square)\\
\equiv\ &\exists\forall\,\exists_2\exists_3\,\square\\
\equiv\ &\exists\forall\exists\square
\end{aligned}$$

A.3 The description of BCL is Σ_5

First, let's treat the statement "$x \in L_i$":

$$\begin{aligned} & x \in L_i && (9) \\ \equiv\ & \exists y \exists z (T(1, \phi_e(i), y, z) \wedge U(z) = x) \\ \equiv\ & \exists \square \end{aligned}$$

Now, the separate conjuncts of the formula $A = A_1 \wedge A_2 \wedge A_3$ are analysed:

$$\begin{aligned} & \forall x \exists s \forall s' > s\big(f_{s'}(x) = f_s(x)\big) && (A_1) \\ \equiv\ & \forall \exists \forall \square \end{aligned}$$

$$\begin{aligned} & \forall x \forall s \Big(\forall s' > s\big(f_{s'}(s) = f_s(x)\big) \rightarrow f_s(x) \in L_i\Big) && (A_2) \\ \equiv\ & \forall \Big(\forall_1 \square \rightarrow \exists_2 \square\Big) \\ \equiv\ & \forall\ \exists_1 \exists_2\ \square \\ \equiv\ & \forall \exists \square \end{aligned}$$

$$\begin{aligned} & \forall x \Bigg(x \in L_i \rightarrow \exists y \exists s \Big(f_s(y) = x \wedge \forall s' > s\big(f_{s'}(y) = f_s(y)\big)\Big)\Bigg) && (A_3) \\ \equiv\ & \forall \Bigg(\exists_1 \square \rightarrow \exists \Big(\square \wedge \forall_2 \square\Big)\Bigg) \\ \equiv\ & \forall\ \forall_1 \Bigg(\square \rightarrow \exists_3\ \forall_2\ \square\Bigg) \\ \equiv\ & \forall\ \exists_3 \Bigg(\square \rightarrow \forall_2 \square\Bigg) \\ \equiv\ & \forall \exists\ \forall_2\ \square \\ \equiv\ & \forall \exists \forall \square \end{aligned}$$

The formula A as a whole is then of the form

$$\begin{aligned} & A_1 \wedge A_2 \wedge A_3 && (A) \\ \equiv\ & \forall_1 \exists \square \wedge \forall_2 \exists \forall \square \wedge \forall_3 \exists \forall \square \\ \equiv\ & \forall_1 \forall_2 \forall_3\ (\exists_4 \square \wedge \exists_5 \forall \square \wedge \exists_6 \forall \square) \\ \equiv\ & \forall\ \exists_4 \exists_5 \exists_6\ (\square \wedge \forall_1 \square \wedge \forall_2 \square) \\ \equiv\ & \forall \exists\ \forall_1 \forall_2\ \square \\ \equiv\ & \forall \exists \forall \square \end{aligned}$$

The other two subformulas of the describing formula are treated next:

$$\exists n \forall s \forall n' > n \Big(\forall s' > s \big(f_{s'} \restriction n' = f_s \restriction n' \big) \to \big(W_{M_k(f_s \restriction n)} = W_{M_k(f_s \restriction n')} \big) \Big) \tag{10}$$

$\equiv \exists\forall(\forall\Box \to \forall_1\exists\Box)$

$\equiv \exists\forall\, \forall_1\, (\forall_2\Box \to \exists_3\Box)$

$\equiv \exists\forall\, \exists_2\exists_3\, \Box$

$\equiv \exists\forall\exists\Box$

$$\forall n \forall s \Bigg(\forall n' > n \forall s' > s \Big(\forall s'' > s' (f_{s''} \restriction n' = f_{s'} \restriction n') \to W_{M_k(f_{s'} \restriction n')} = W_{M_k(f_s \restriction n)} \Big) \\ \to W_{M_k(f_s \restriction n)} = L_i \Bigg) \tag{11}$$

$\equiv \forall\Big(\forall(\forall\Box \to \forall_1\exists\Box) \to \forall_2\exists\Box\Big)$

$\equiv \forall\, \forall_2\, \Big(\, \forall_1\, (\forall_3\Box\exists_4\Box) \to \exists\Box\Big)$

$\equiv \forall\Big(\forall_1\, \exists_3\exists_4\, \Box \to \exists_2\Box\Big)$

$\equiv \forall\, \exists_1\exists_2\, \Big(\exists_3\Box \to \Box\Big)$

$\equiv \forall\exists\, \forall_3\, \Box$

$\equiv \forall\exists\forall\Box$

Finally, the resulting formula is of the form

$\exists\forall\Big(\forall\exists\forall\Box \to (\exists\forall\exists\Box \wedge \forall_1\exists\forall\Box)\Big)$

$\equiv \exists\forall\Big(\forall\exists\forall\Box \to \, \forall_1\, (\exists_2\forall\exists\Box \wedge \exists_3\forall\Box)\Big)$

$\equiv \exists\forall\, \forall_1\, \Big(\forall_4\exists\forall\Box \to \, \exists_2\exists_3\, (\forall_5\exists\Box \wedge \forall_6\Box)\Big)$

$\equiv \exists\forall\, \exists_4\exists_2\exists_3\, \Big(\exists_1\forall\Box \wedge \forall_5\forall_6(\exists_7\Box \wedge \Box)\Big)$

$\equiv \exists\forall\exists\, \forall_1\forall_5\forall_6\, \Big(\forall_2\Box \to \, \exists_7\, \Box\Big)$

$\equiv \exists\forall\exists\forall\, \exists_2\exists_7\, \Box$

$\equiv \exists\forall\exists\forall\exists\Box$

A.4 The description of EXL* is Σ_5

Some of the subformulas used in the describing formula of EXL* were already treated in Appendix A.1. Only the formula B and the expression $W_{M_k(\phi_a \restriction n)} \triangle L_i = W_{h(l)}$, which is part of B, need to be analysed:

$$W_{M_k(\phi_a \restriction n)} \triangle L_i = W_{h(l)} \tag{12}$$

$$\begin{aligned}
&\equiv \forall x\Big(\big(x \in W_{M_k(\phi_a \restriction n)} \leftrightarrow x \notin L_i\big) \leftrightarrow x \in W_{h(l)}\Big)\\
&\equiv \forall x\Big(\big(\exists y T(1, M_k(\phi_a \restriction n), x, y) \leftrightarrow \forall z \neg T(1, \phi_e(i), x, z)\big) \leftrightarrow \exists w T(1, h(l), x, w)\Big)\\
&\equiv \forall\Big(\big(\exists\Box \leftrightarrow \forall\Box\big) \leftrightarrow \exists\Box\Big)\\
&\equiv \forall\Big(\big((\exists_1\Box \rightarrow \forall_2\Box) \wedge (\forall_3\Box \rightarrow \exists_4\Box)\big) \leftrightarrow \exists\Box\Big)\\
&\equiv \forall\Big(\big(\forall_1\forall_2\Box \wedge \exists_3\exists_4\Box\big) \leftrightarrow \exists\Box\Big)\\
&\equiv \forall\Big(\big((\forall\Box \wedge \exists_1\Box) \rightarrow \exists\Box\big) \wedge \big(\exists\Box \rightarrow (\forall_2\Box \wedge \exists\Box)\big)\Big)\\
&\equiv \forall\Big(\big(\exists_1(\forall_3\Box \wedge \Box) \rightarrow \exists\Box\big) \wedge \big(\exists\Box \rightarrow \forall_2(\Box \wedge \exists_4\Box)\big)\Big)\\
&\equiv \forall\Big(\big(\exists_1\forall_3\Box \rightarrow \exists\Box\big) \wedge \big(\exists_5\Box \rightarrow \forall_2\exists_4\Box\big)\Big)\\
&\equiv \forall\Big(\forall_1\big(\forall_3\Box \rightarrow \exists_6\Box\big) \wedge \forall_5\forall_2\big(\Box \rightarrow \exists_4\Box\big)\Big)\\
&\equiv \forall\Big(\forall_1\exists_3\exists_6\Box \wedge \forall_5\exists_4\Box\Big)\\
&\equiv \forall\forall_1\forall_5\Big(\exists_3\Box \wedge \exists_4\Box\Big)\\
&\equiv \forall\exists_3\exists_4\Box\\
&\equiv \forall\exists\Box
\end{aligned}$$

– –

$$\exists l \forall n\Big(\forall m\Big(m > n \rightarrow M_k(\phi_a \restriction m) = M_k(\phi_a \restriction n)\Big) \rightarrow \Big(W_{M_k(\phi_a \restriction n)} \triangle L_i = W_{h(l)}\Big)\Big) \tag{B}$$

$$\begin{aligned}
&\equiv \exists\forall(\forall\Box \rightarrow \forall_1\exists\Box)\\
&\equiv \exists\forall\forall_1(\forall_2\Box \rightarrow \exists_3\Box)\\
&\equiv \exists\forall\exists_2\exists_3\Box\\
&\equiv \exists\forall\exists\Box
\end{aligned}$$

The formula describing EXL*-learnability is Σ_5:

$$\begin{aligned}
&\exists\forall\exists\Big(\forall\exists\square \wedge \Big((\forall_1\exists\square \wedge \forall_2\exists\square) \to (\exists_3\forall\square \wedge \exists_4\forall\exists\square)\Big)\Big)\\
\equiv\ &\exists\forall\exists\Big(\forall\exists\square \wedge \Big(\forall_1\forall_2\,(\exists\square \wedge \exists\square) \to \exists_3\exists_4\,(\forall\square \wedge \forall\exists\square)\Big)\Big)\\
\equiv\ &\exists\forall\exists\Big(\forall\exists\square \wedge \exists_1\exists_2\exists_3\exists_4\Big((\exists_5\square \wedge \exists_6\square) \to (\forall_7\square \wedge \forall_8\exists\square)\Big)\Big)\\
\equiv\ &\exists\forall\exists\,\exists_1\exists_2\exists_3\exists_4\Big(\forall\exists\square \wedge \Big(\exists_5\exists_6\,\square \to \forall_7\forall_8\,(\square \wedge \exists\square)\Big)\Big)\\
\equiv\ &\exists\forall\exists\Big(\forall_1\exists\square \wedge \forall_5\forall_6\forall_7\forall_8\Big(\square \to (\square \wedge \exists_2\square)\Big)\Big)\\
\equiv\ &\exists\forall\exists\,\forall_1\forall_5\forall_6\forall_7\forall_8\Big(\exists\square \wedge \Big(\square \to \exists_2\,\square\Big)\Big)\\
\equiv\ &\exists\forall\exists\forall\Big(\exists_1\square \wedge \exists_2\,\square\Big)\\
\equiv\ &\exists\forall\exists\forall\,\exists_1\exists_2\,\square\\
\equiv\ &\exists\forall\exists\forall\exists\square
\end{aligned}$$

B Notations used

$\mathbb{N}$	$\{0, 1, 2, 3, ...\}$
$[a, b]$	interval $\{a, a + 1, ..., b\}$
$\lfloor x \rfloor$	floor of x
$\lvert A \rvert$	cardinality of A
$\overline{A}$	complement of A
$A \cup B$	union of A and B
$A \cap B$	intersection of A and B
$A \backslash B$	difference between A and B
$A \triangle B$	symmetric difference between A and B (p. 12)
$A \subseteq B$	A is a subset of B
$A \subsetneq B$	A is a strict subset of B
$A \leq_m B$	A is m-reducible to B (p. 15)
ϕ_e	function with index e (p. 6)
M_e	Turing machine with index e (p. 6)
W_e	domain of the function ϕ_e (p. 6)
$g \circ h$	composition of g and h
$\mu x.\phi$	the least x such that ϕ (p. 4)
$T(m, e, x, y)$	Kleene T-predicate (p. 6)
$U(y)$	output function (p. 6)
$\langle \vec{x} \rangle$	code for $\vec{x}$ (p. 8)
$(x)_i$	i^{th} element of tuple coded by x (p. 8)
$\text{lh}(x)$	length of tuple coded by x (p. 8)
$x \star y$	concatenation (p. 8)
$\alpha \prec \beta$	α is an initial segment of β
$f \restriction (t + 1)$	$\langle f(0), f(1), ..., f(t) \rangle$ (p. 9)
$(a_n)_{n \in \mathbb{N}}$	sequence $a_0, a_1, a_2, ...$