Making Solomonoff Induction Effective or You Can Learn What You Can Bound

Jörg Zimmermann and Armin B. Cremers

Institute of Computer Science,
University of Bonn, Germany
{jz, abc}@iai.uni-bonn.de

Abstract. The notion of effective learnability is analyzed by relating it to the proof-theoretic strength of an axiom system which is used to derive totality proofs for recursive functions. The main result, the generator-predictor theorem, states that an infinite sequence of bits is learnable if the axiom system proves the totality of a recursive function which dominates the time function of the bit sequence generating process.
This result establishes a tight connection between learnability and provability, thus reducing the question of what can be effectively learned to the foundational questions of mathematics with regard to set existence axioms. Results of reverse mathematics are used to illustrate the implications of the generator-predictor theorem by connecting a hierarchy of axiom systems with increasing logical strength to fast growing functions. Our results are discussed in the context of the probabilistic universal induction framework pioneered by R. J. Solomonoff, showing how the integration of a proof system into the learning process leads to naturally defined effective instances of Solomonoff induction. Finally, we analyze the problem of effective learning in a framework where the time scales of the generator and the predictor are coupled, leading to a surprising conclusion.

1 Introduction

An effective learning system is a system which can be fully specified by a program on a universal Turing machine. In its most general form, this program transforms a stream of percepts generated by an environment (later called the generator) into a stream of actions possibly changing this environment. Via this sensomotoric loop the system is embedded into its environment, about which a priori nothing is known [7]. A more specific notion of learning is defined as the process which translates the stream of percepts into predictions for future percepts. These predictions can then be used for choosing actions. The analysis of the "design space" for effective learning systems leads to three major questions:

1. How should a learning system represent and process uncertainty, or, what is the proper inductive logic?
2. What set of possible models of the environment should the system consider?

3. How to relate the explanatory power of a model to its complexity?

In the long run, the learning system should be able to detect as many regularities in its percept stream as possible, while dealing sensibly with the inherent uncertainty of predictions based on a finite amount of data. The first question regarding the representation and processing of uncertainty is in itself a current area of research, where a plethora of different approaches are discussed [5]. An attempt to find a unifying perspective on these approaches has been made in [15]. However, this question is not the focus of this contribution, so we will discuss only a probabilistic learning framework, which is essentially a dynamic extension of Bayesian inference. Here we focus on the second and third question.

The definition of proof-driven learning systems, combining search in proof and program space, was inspired by an algorithm solving all well-defined problems as fast as possible (save for a factor of 5 and additive terms) introduced by M. Hutter in [6]. We apply this idea in the context of learning in the limit and discuss in more detail the dependence of concepts like well-definedness of programs or effective learnability on the proof-theoretic strength of the employed background theory, thus establishing an explicit link between effective learnability and the foundational mathematical questions treated in proof theory and ordinal analysis [10]. Additionally, our results emphasize the fact that provability is not an absolute concept. The theoretical limits of effective learnability and computable approximations to Solomonoff induction are intensively discussed in various publications investigating different aspects of this topic, see, for example, [8] and the references therein, but to our knowledge none has made the connection between proof-theoretic strength, time complexity and effective learnability as explicit as it is stated in our main result, the generator-predictor theorem. The first learning in the limit framework, wich does not consider the uncertainty of predictions, was introduced by E. M. Gold [3]. A discussion of our results in the context of this framework can be found in [16].

2 The Generator Space

A clarification of the concept of learnability has to consider the following question: What is the set of possible generators for observed events? To answer this question, we will explore the notion of "all possible generators" from a mathematical and computational point of view, and discuss the question of effective learnability in the context of such generic model spaces. The last sentence uses intentionally the plural form of model space, because we will see that the notion of "all possible models" cannot be defined in an absolute sense, but only with regard to a reference proof system. This dependence will lead to the establishment of a relationship between the *time* complexity of the percept-generating environment and the *logical* complexity of an effective learning system, thus shedding new light on the undecidability of a general approach to induction developed by R. J. Solomonoff in the 1960s [12, 13, 9].

2.1 Algorithmic Ontology: Programs as Generators

Ontology is a part of philosophy which tries to define what exists, or, more specifically, what *possibly* could exist. In the realm of mathematics, this question leads to the set existence problem, which is (partially) answered by various set theories, most commonly by using the axiom system **ZFC**, Zermelo-Fraenkel set theory with the axiom of choice. But in the realm of computer science, existence has to be *effective* existence, i.e. the domain of interest and its operations must have effective representations.

For this reason the objects we consider are programs executed by a fixed universal Turing machine U having a one-way read-only input tape, some work tapes, and a one-way write-only output tape (such Turing machines are called monotone), which will be the reference machine for all what follows. The choice of the specific universal Turing machine has only a constant factor as effect on the space complexity of a program and at most a logarithmic factor as effect on the time complexity [1]. These effects are small enough to be neglected in the following foundational considerations, but in the context of alternative models of computation, like cellular automata on infinite grids, the choice of the reference machine may become an important issue. The program strings are chosen to be prefix-free, i.e. no program string is the prefix of another program string. This is advantageous from a coding point of view (enabling, for example, the application of Kraft's inequality $\sum_p 2^{-length(p)} \leq 1$), and does not restrict universality [9].

A program p (represented as finite binary string) is a generator of a possible world, if it outputs an infinite stream of bits when executed by U. Unfortunately, it is not decidable whether a given program p has this well-definedness property. This is the reason why the general approach to induction introduced by R. J. Solomonoff is incomputable: the inference process uses the whole set of programs (program space) as possible generators, even the programs which are not well-defined in the above sense.

This results in the following dilemma: either one restricts the model space to a decidable set of well-defined programs, which leads to an effective inference process but ignores possibly meaningful programs, or one keeps all well-defined programs, but at the price of necessarily keeping ill-defined programs as well, risking the incomputability of the inference process. However, in the following we propose an approach which tries to mitigate this dilemma by reducing the question of learnability to the question of provability.

3 Learning Systems

Here we introduce the notion of a *probabilistic learning system*, which takes a finite string of observed bits (the percept string) as input and produces a probabilistic prediction for the next bit as output:

Definition 1. *A* probabilistic learning system *is a function*

$$\Lambda : \{0,1\}^* \times \{0,1\} \to [0,1]_{\mathbf{Q}}, \quad \text{with } \Lambda(x,0) + \Lambda(x,1) = 1 \text{ for all } x \in \{0,1\}^*.$$

Λ is an *effective* probabilistic learning system if Λ is a total recursive function. We use rational numbers as probability values, because real numbers cannot be used directly in a context of computability questions, but have to be dealt with by effective approximations [14]. This would increase the complexity of our definitions significantly, and is not necessary for a first understanding of the fundamental relationship between learnability and provability. Also we treat only deterministic generators leading to one definite observable bit sequence. A generalization of our results to real valued probabilities and probabilistic generators should be possible, but is open to future research.

Next we extend the prediction horizon of Λ by feeding it with its own predictions. This leads to a learning system $\Lambda^{(k)}$ which makes probabilistic predictions for the next k bits (xy is the concatenation of string x and y):

$\Lambda^{(1)} = \Lambda,$
$\Lambda^{(k+1)}(x, y1) = \Lambda^{(k)}(x, y) \cdot \Lambda(xy, 1), \quad x \in \{0,1\}^*, y \in \{0,1\}^k,$
$\Lambda^{(k+1)}(x, y0) = \Lambda^{(k)}(x, y) \cdot \Lambda(xy, 0).$

Finally, we define the learnability of an infinite bit sequence s ($s_{i:j}$ is the subsequence of s starting with bit i and ending with bit j):

Definition 2. *An infinite bit sequence s is learnable in the limit by the probabilistic learning system Λ, if for all $\epsilon > 0$ there is an n_0 so that for all $n \geq n_0$ and all $k \geq 1$:*

$$\Lambda^{(k)}(s_{1:n}, s_{n+1:n+k}) > 1 - \epsilon.$$

4 Σ-driven Learning Systems

We now introduce the learning systems we will use to investigate the notion of effective learnability. But first we need the following definitions specifying the nature of background theories available to these learning systems.

Definition 3. *A* logic frame *is a 5-tuple $\Sigma = (S, F, \models, \vdash, \Phi)$, where elements of S are called the structures of Σ, elements of F are called the sentences of Σ, the relation $\models \subseteq S \times F$ is called the satisfaction relation of Σ, $\vdash : 2^F \to 2^F$ is the deduction system of Σ, and $\Phi \subseteq F$ is the core axiom system of Σ.*

A deduction system is *sound* if for all $\Psi \subseteq F$ it holds: if $\Psi \vdash \psi$, then for all $M \in S$: if $M \models \Psi$, then $M \models \psi$. Next we define admissibility for logic frames:

Definition 4. *Let $\Sigma = (S, F, \models, \vdash, \Phi)$ be a logic frame. Σ is* admissible *if $\vdash$ is sound, Φ is enumerable, $\vdash$ effectively maps enumerable sets of axioms to enumerable sets of logical consequences, and for all recursive functions f there is a sentence $\phi_{tot}(f) \in F$ with the following property: if $\Phi \vdash \phi_{tot}(f)$, then f is a total recursive function.*

Note that this definition only fixes the meaning of $\phi_{tot}(f)$ when it is derivable from Φ. A pathological definition like $\phi_{tot}(f) = FALSE$ for all f is consistent with the above definition of a logic frame, but would imply that such a logic frame could not prove the totality of any recursive function f.

Let $\phi_1, \ldots, \phi_n$ be the first n sentences enumerated by the deduction system of an admissible logic frame Σ, then the set $Tot_n(\Sigma)$ is defined as follows:

$$f \in Tot_n(\Sigma) \quad \text{iff} \quad \phi_{tot}(f) \in \{\phi_1, \ldots, \phi_n\}.$$

So $Tot_n(\Sigma)$ contains the recursive functions which admit a totality proof within the first n sentences enumerated by the deduction system of Σ.

Now let $g_n(m) = max(\{f(m) | f \in Tot_n(\Sigma)\})$ and $g(n) = g_n(n)$. g is called the *guard function* wrt. Σ. The maximum over an empty set is defined as 0. Note that the maximum is taken over a finite set of total recursive functions, so the guard function is a total recursive function, too. The guard function g will play a central role as a scheduler ensuring the effectiveness of learning systems.

Let Σ be an admissible logic frame, then the *Σ-driven probabilistic learning system* $\Lambda(\Sigma)$ is defined as follows: $\Lambda(\Sigma)$ uses a dynamic model space and dynamic prior probabilities and considers at inference step n only the part of the program space given by the programs with length at most n. Because n grows unboundedly, eventually every program will be part of the model space, but at every instant of time the model space is finite. This assumption implies that the posterior distributions and the probabilistic predictions can be computed exactly and all involved probabilities are rational numbers. $\Lambda(\Sigma)$ also manages three labels for programs in addition to their current probabilities, in contrast to classical Baysian inference: “candidate”, “suspended”, and “discarded”. A program which is added to the model space initially gets the label “candidate”, all other programs keep their label from the previous inference step. After the nth bit has been observed, $\Lambda(\Sigma)$ executes the following steps:

Input: the nth observed bit.
Output: a probabilistic prediction for the next bit.

1. Derive ϕ_n using the deduction system of Σ.
2. Compute $g(n)$.
3. Initialize $\mu_* = 0$. This variable accumulates preliminary posterior probability mass and is needed to normalize the posterior distribution.
4. Start the enumeration of all programs p with $length(p) \leq n$.
5. If a program p is labeled “discarded”, it has already probability 0 and nothing has to be done.
6. If a program p is labeled “suspended” or “candidate”, evaluate program p till it outputs n bits or its step function reaches $g(n)$.
7. If p has generated a bit and it is the observed one, then label p as “candidate”. Assign p a preliminary posterior probability of $2^{-(length(p)+switch(p,n))}$, where $switch(p, n)$ counts the number of switches p has experienced from “suspended” status back to “candidate” status up to now. A higher number of switches is translated retroactively into a lower prior probability. Add the preliminary posterior probability to μ_*

8. If the generated bit is not the observed one, label p as "discarded" and set its posterior probability to 0.
9. If p has reached the time limit specified by the guard function g, then label p as "suspended" and set its posterior probability to 0.
10. Continue the enumeration of programs.
11. If the enumeration of all programs p with $length(p) \leq n$ is completed and no program has the label "candidate", return $(\frac{1}{2}, \frac{1}{2})$ as probabilistic prediction for the next bit. Exit this inference step.
12. Rescale the preliminary posterior probabilities of all models labeled "candidate" by $1/\mu_*$, resulting in a normalized posterior distribution on the currently considered model space. Use the guarded versions $p \restriction g$ of the programs p labeled "candidate" for computing the probabilistic prediction for the next bit. ($p \restriction g$, *p guarded by g*, is the time limited version of p: if the step function (the function which just counts the transitions made by p) of $p(m)$ exceeds $g(m)$, then the computation of p is terminated and 0 is returned, else $p(m)$ is returned).

This finishes our definition of Σ-driven probabilistic learning systems. Next we will formulate and prove a theorem characterizing their learning capabilities.

5 Generator-Predictor Theorem

If the generator functions (see below) of all programs p generating a bit sequence s grow so fast that they are not dominated by a provably total recursive function, then it is indistinguishable, even in the limit, from a non-effective bit sequence. So the quest for making Solomonoff induction effective can be reduced to the concept of provably total recursive functions. But which functions are provably total recursive and which are not? The answer is: it depends. It depends on the background theory or logic frame which is accepted for the construction of totality proofs. At this point we have reduced the problem of universal induction to a logical parameter, the logic frame Σ to which the learning system can refer. The next step is to relate the proof strength of a logic frame Σ to the time complexity of a program p which generates the bit stream observed by our learning system. This will yield a natural characterization of the learnable bit sequences relative to Σ.

Before we can state this relationship as a theorem, we need the notion of the *generator time function*, generator function for short, of a program p:

Definition 5. *The* generator time function $G_p^{(U)} : \mathbf{N} \to \mathbf{N} \cup \{\infty\}$ *of a program p wrt. the universal reference machine U assigns every $n \in \mathbf{N}$ the number of transitions needed to generate the first n bits by the reference machine U executing p. If n_0 is the smallest number for which p does not generate a new bit, then $G_p^{(U)}(n) = \infty$ for all $n \geq n_0$.*

In the following we will drop the superscript (U) because we are working only with one reference machine.

In general there are several programs generating the same observable bit sequence. So one can not hope to learn exactly the program p which generates the observed bit sequence, but only a program p' which is *observation equivalent*. The equivalence class of programs corresponding to an infinite bit sequence s we will denote by $[s]$. Now we have introduced all the notions and concepts we need in order to state our main result:

Generator-Predictor Theorem: Let Σ be an admissible logic frame and s an infinite bit sequence. s is learnable by the effective probabilistic learning system $\Lambda(\Sigma)$, if Σ proves the totality of a recursive dominator of a generator function G_p for at least one program $p \in [s]$.

Proof: Let $L_\Sigma(s) = \{p | p \in [s]$ and Σ proves the totality of a recursive dominator of the generator function of $p\}$. So $L_\Sigma(s)$ contains the programs which can be eventually used by $\Lambda(\Sigma)$ as perfect predictors. If $L_\Sigma(s)$ is empty, the theorem makes no statement about the learnability of s, so lets assume that $L_\Sigma(s)$ contains at least one element. We will show that the sum of the posterior probabilities of the programs in $L_\Sigma(s)$ will converge to 1 as the number of observed bits increases. This will be conducted in two steps. Let $\alpha(p,n)$ be the preliminary posterior probability (i.e., the posterior probability assigned to p before normalization) of program p in the nth inference step (programs not contained in the nth model space are considered as having a preliminary posterior probability of 0), $\alpha(n)$ be the sum of the preliminary posterior probabilities of all programs in $L_\Sigma(s)$ and $\bar{\alpha}(n)$ be the sum of the preliminary posterior probabilities of all programs not in $L_\Sigma(s)$. Note that $\alpha(n)+\bar{\alpha}(n) \neq 1$, because preliminary and not final posterior probabilities are considered. First we will show the existence of a number n_1 so that $\alpha(n) \geq c$ for some constant $c > 0$ and for all $n \geq n_1$. And second, we will see that $\bar{\alpha}(n)$ converges to 0 as the number of inference steps n goes to infinity. This implies that the normalized value of $\alpha(n)$ has to go to 1 as n goes to infinity:

$$\alpha^{norm}(n) = \frac{\alpha(n)}{\alpha(n)+\bar{\alpha}(n)} = \frac{1}{1+\frac{\bar{\alpha}(n)}{\alpha(n)}} \quad \underset{n\to\infty}{\longrightarrow} \quad 1$$

So both steps together will establish the generator-predictor theorem.

If $p \in L_\Sigma(s)$, then there is a number n_0 so that the guard function g majorizes G_p for all $n \geq n_0$. Let $n_1 = \max(n_0, \mathit{length}(p))$. Then p is part of the model space for all $n \geq n_1$ and it is not discarded and not suspended. Its preliminary posterior probability is $2^{-(length(p)+switch(p,n_1))}$. This number does not change anymore for $n \geq n_1$ (no new switches), thus $c = 2^{-(length(p)+switch(p,n_1))} > 0$ is a lower bound for $\alpha(n)$ for all $n \geq n_1$. This completes the first step.

Kraft's inequality for prefix codes implies $\sum_{p \notin L_\Sigma(s)} 2^{-length(p)} \leq 1$. Thus for all $\epsilon > 0$ there is a k_0 with $\sum_{p \notin L_\Sigma(s), length(p) > k} 2^{-length(p)} < \epsilon$ for all $k \geq k_0$, because for a convergent sum the partial sums $\sum_{p \notin L_\Sigma(s), length(p) \leq k} 2^{-length(p)}$ converge to the limit. Now choose n_2 so that $\sum_{p \notin L_\Sigma(s), length(p) > n} 2^{-length(p)} < \epsilon/2$ for all $n \geq n_2$, and n_3 so that $\sum_{p \notin L_\Sigma(s), length(p) \leq n_2} \alpha(p,n) < \epsilon/2$ for all

$n \geq n_3$. n_3 exists, because for a fixed n_2 there are only finitely many summands $\alpha(p,n)$ contributing to the sum, each dropping to 0 (discarded or suspended forever) or converging to 0 (number of switches increases unboundedly) as n goes to infinity. Then for all $n \geq n_3$ we have:

$$\bar{\alpha}(n) = \sum_{p \notin L_\Sigma(s)} \alpha(p,n) = \sum_{\substack{p \notin L_\Sigma(s) \\ length(p) \leq n_2}} \alpha(p,n) \quad + \sum_{\substack{p \notin L_\Sigma(s) \\ length(p) > n_2}} \alpha(p,n)$$

$$\leq \sum_{\substack{p \notin L_\Sigma(s) \\ length(p) \leq n_2}} \alpha(p,n) \quad + \sum_{\substack{p \notin L_\Sigma(s) \\ length(p) > n_2}} 2^{-length(p)} \quad < \quad \epsilon/2 + \epsilon/2 \quad = \quad \epsilon$$

This finishes the second step and thus the proof of the generator-predictor theorem.

□

For example, if $\Sigma_{PA} = (FOL, PA)$ (First order logic Peano Arithmetic), then every program with a generator function which is dominated by a provably total recursive function wrt. PA can be learned by $\Lambda(\Sigma_{PA})$. As the Ackermann-function is provably total in PA, this is already a pretty large set, and PA allows totality proofs of functions which grow much faster than the Ackermann-function.

The Generator-Predictor theorem states that for learning a bit sequence s it is enough to prove the totality of a recursive dominator of the generator function of a program $p \in [s]$, it is not necessary to prove the totality of p itself. However, one can show that whenever there is a $p \in [s]$ so that the logic frame Σ can prove the totality of a recursive dominator of G_p, then there is a $q \in [s]$ for which Σ can prove totality, provided the logic frame satisfies some weak closure conditions [16].

6 Learnability and Reverse Mathematics

Reverse mathematics is a program in mathematical logic that seeks to determine which set existence axioms are required to prove theorems of mathematics in order to classify these theorems according to the strength of existence axioms necessary to derive them. The program was founded in the 1970s by H. Friedman [2]. Maybe the most remarkable fact of reverse mathematics is that many theorems of classical mathematics fall into five large equivalence classes, consisting of provably equivalent theorems. This leads to five standard axiom systems, which are linearly ordered according to their proof-theoretic strength. S. G. Simpson has named them the “Big Five” [11], starting with a system called RCA_0 (Recursive Comprehension Axiom), followed by WKL_0 (Weak König's Lemma), ACA_0 (Arithmetical Comprehension Axiom), ATR_0 (Arithmetical Transfinite Recursion), and $\Pi^1_1 - CA_0$ (Π^1_1-Comprehension Axiom).

One can show that the functions which admit totality proofs by RCA_0 are exactly the primitive recursive functions. This implies that a bit stream $s_{Ackermann}$

which has a generating program with a generator function growing like the Ackermann-function – a function known not to be primitive recursive – but no generating program with a generator function dominated by a primitive recursive function, can not be learned by $\Lambda(RCA_0)$. In terms of totality proofs WKL_0 adds no additional proof-strength to RCA_0, but the next system, ACA_0, can prove the totality of the Ackermann-function, so $\Lambda(ACA_0)$ can learn $s_{Ackermann}$ and is therefore a stronger learning system than $\Lambda(RCA_0)$. But there are total recursive functions which have no totality proof in ACA_0 (which is equivalent to Peano Arithmetic in this regard), for example the Goodstein-function [4]. This can be continued even beyond $\Pi^1_1 - CA_0$, leading to the foundational questions within mathematics concerning the introduction of ever stronger set existence axioms. Here it suffices to note that these examples are a good illustration of the fact that increased proof-theoretic strength translates into stronger learning systems.

7 Synchronous Learning Frameworks

A closer look on real world incremental learning situations, where both, the environment and the learning system, are not suspended while the other one is performing its transitions resp. computations, leads to the following notion of *synchrony* of a bit sequence s:

Definition 6. *s is* synchronous *if* $\limsup_{n\to\infty} \frac{G_p(n)}{n} < \infty$ *for at least one* $p \in [s]$.

Synchrony entails that the time scales of the learning system and the environment are coupled, that they cannot ultimately drift apart. As long as one not assumes a malicious environment, synchrony seems to be a natural property. Such a setting for learning could be called a synchronous learning framework, in contrast to the above considered learning frameworks, which could be classified as asynchronous.

In order to learn in the case of synchrony, it suffices to prove that n^2 is total, because n^2 is a dominator of every generator function satisfying the synchrony condition. Thus, a much weaker background theory than, e.g. Peano Arithmetic would suffice for an effective learning system to learn all synchronously generated bit sequences. In fact, because RCA_0 – the weakest of the five standard axiom systems considered in reverse mathematics – proves the totality of all primitive recursive functions, $\Lambda(RCA_0)$ is a perfect learning system in a synchronous world.

8 Conclusion

We argue that the generator-predictor theorem establishes a natural perspective on the effective core of Solomonoff induction by shedding new light on the cause of the incomputability of the non-relativized Solomonoff induction, instead of directly introducing specific resource constraints in order to achieve computability, like this is done, for example, in [7] for the AIξ learning system. This shifts the questions related to learnability to questions related to provability, and therefore into the realm of the foundations of mathematics.

The problem of universal induction in the synchronous learning framework, however, is intrinsically effective, and the focus of future research in a synchronous framework can be on *efficiency* questions. In fact, the source of incomputability in the asynchronous learning framework can be traced back to the fact that the learning system does not know how much time the generator process has "invested" in order to produce the next bit. An extension of a bit sequence s by inserting "clock signals" (coding a clock signal by "00" and output bits by "10" and "11") marking the passing of time would transform every sequence s into a synchronous one, thus eliminating the incomputability of Solomonoff induction. So the synchronous learning framework seems to be perfectly suited for studying the problem of universal induction from a computational point of view.

References

1. S. Arora and B. Barak. *Complexity Theory: A Modern Approach.* Cambridge University Press, 2009.
2. H. Friedman. Some systems of second order arithmetic and their use. In *Proceedings of the International Congress of Mathematicians (Vancouver, B.C.)*, volume 1, pages 235–242. Canad. Math. Congress, Montreal, Que., 1974.
3. E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
4. R. Goodstein. On the restricted ordinal theorem. *Journal of Symbolic Logic*, 9:33–41, 1944.
5. F. Huber and C. Schmidt-Petri, editors. *Degrees of Belief.* Springer, 2009.
6. Marcus Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(3):431–443, 2002.
7. Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, 2005.
8. S. Legg. Is there an elegant universal theory of prediction? In *Proc. 17th International Conf. on Algorithmic Learning Theory (ALT06)*, pages 274–287, 2006.
9. Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications (3. ed.).* Graduate texts in computer science. Springer, 2008.
10. M. Rathjen. The art of ordinal analysis. In *Proceedings of the International Congress of Mathematicians*, pages 45–69. Eur. Math. Soc., 2006.
11. S. G. Simpson. *Subsystems of Second Order Arithmetic (2. ed.).* Cambridge University Press, 2009.
12. R. Solomonoff. A formal theory of inductive inference, part I. *Information and Control*, 7(1):1–22, 1964.
13. R. Solomonoff. A formal theory of inductive inference, part II. *Information and Control*, 7(2):224–254, 1964.
14. K. Weihrauch. *Computable analysis.* Springer, 2000.
15. J. Zimmermann and A. B. Cremers. The Quest for Uncertainty. In Cristian S. Calude, Grzegorz Rozenberg, and Arto Salomaa, editors, *Rainbow of Computer Science*, volume 6570 of *Lecture Notes in Computer Science.* Springer, 2011.
16. J. Zimmermann and A. B. Cremers. Proof-driven learning systems. *http://www.iai.uni-bonn.de/~jz/pls.pdf*, 2012.