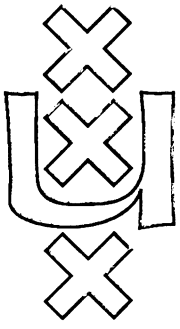
Institute for Language, Logic and Information

A THEORY OF LEARNING SIMPLE CONCEPTS UNDER SIMPLE DISTRIBUTIONS AND AVERAGE CASE COMPLEXITY FOR THE UNIVERSAL DISTRIBUTION

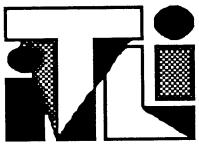
(PRELIMINARY VERSION)

Ming Li
Paul M.B. Vitanyi

ITLI Prepublication Series
for Computation and Complexity Theory CT-89-07



University of Amsterdam



Instituut voor Taal, Logica en Informatie
Institute for Language, Logic and Information

Faculteit der Wiskunde en Informatica
(Department of Mathematics and Computer Science)
Plantage Muidergracht 24
1018TV Amsterdam

Faculteit der Wijsbegeerte
(Department of Philosophy)
Nieuwe Doelenstraat 15
1012CP Amsterdam

A THEORY OF LEARNING SIMPLE CONCEPTS UNDER SIMPLE DISTRIBUTIONS AND AVERAGE CASE COMPLEXITY FOR THE UNIVERSAL DISTRIBUTION

(PRELIMINARY VERSION)

Ming Li
Computer Science Department, University of Waterloo
Paul M.B. Vitanyi
Department of Mathematics and Computer Science, University of Amsterdam
Centre for Mathematics and Computer Science, Amsterdam

Received October 1989

A Theory of Learning Simple Concepts Under Simple Distributions and Average Case Complexity for the Universal Distribution (Preliminary Version)

*Ming Li**

Computer Science Department, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1.
(mli@water.waterloo.edu)

Paul M.B. Vitanyi

Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ Amsterdam
and
Universiteit van Amsterdam
Faculteit Wiskunde en Informatica
(paulv@cwi.nl)

"The human mind, encouraged by the success of its solutions, becomes conscious of its independence. It evolves from itself alone, often without any appreciable influence from without, by means of logical combination, generalization, specialization, by separating and collecting ideas in fortunate ways, new and fruitful problems, and appears then itself as the real questioner." [D. Hilbert, Mathematical Problems, 1900]

Abstract

We treat two distinct topics which are related through the use of the so-called universal distribution. In the first part we develop a learning theory where 'simple' concepts are easily learnable. In Valiant's learning model, many concepts turn out to be too hard (like NP hard) to learn. Relatively few concept classes were shown to be learnable polynomially. In practice, almost nothing we care to learn appears to be not learnable. To model the intuitive notion of learning more closely, we impose a reasonable restriction on Valiant's model. We assume that learning happens under an *arbitrary simple* distribution, rather than an *arbitrary* distribution as assumed by Valiant [V]. A distribution is simple if it is dominated by a recursively enumerable distribution. Such an assumption appears to be not very restrictive in a practical sense. Most distributions we customarily deal with are recursive or can be approximated or dominated by recursive or recursively enumerable ones, hence they fit our assumption. We systematically develop a general theory of learning under simple distributions. In particular we show a completeness result: one can learn under all simple distributions, iff one can learn under one fixed simple distribution, the 'universal' distribution. We use this completeness result to obtain new learning algorithms and several quite general new learnable classes. These classes are more general than the classes known to be polynomially learnable in Valiant's original model.

In the second, short, part we show that for essentially all algorithms, if the inputs are distributed according to the universal distribution, then the average case complexity is of the same order of magnitude as the worst-case complexity.

Note: A brief version of this paper appears in: *Proceedings 30th Annual IEEE Symposium on Foundations of Computer Science*, held Oct. 30 - Nov. 1, 1989, Research Triangle Park, North Carolina. The first part of this paper was also presented at *2nd Annual ACM Workshop on Computational Learning Theory*, held July 31 - August 2, 1989, Santa Cruz, California.

---

* The first author is supported by the NSERC operating grant OGP0036747. He performed part of this work while at the Computer Science Department, York University, North York, Ontario, Canada.

1. Introduction to Learning Theory Part

Valiant [V] has proposed a learning theory, where one wants to learn a concept in polynomial time, within a certain error, under all distributions on the examples. But many subsequent investigations resulted in negative, hardness, or equivalence results [G2,A2,KLPV,PW,KV,PW1,PV]. There are at least two problems with Valiant's proposal in [V]:

(1) Under all distributions, many concept classes, including some seemingly simple ones, are not known to be polynomially learnable or known not to be polynomially learnable if $NP \neq RP$, although some concept classes are polynomially learnable under some fixed distribution.

(2) In real life situations, it is sometimes impossible to sample according to underlying distributions.

Item (1) is counterintuitive for a proposed theory of machine learning; in fact it shows that Valiant's requirements for learning are too harsh. In practice, we usually do not have to make such a general assumption. For this reason, several authors have proposed to study Valiant learning under fixed distribution [KLPV,N,BI]. Then some previously (polynomially) unlearnable classes become learnable. For instance, the class of μ-formulae is polynomially learnable under the uniform distribution. However, the assumption of *any special* distribution is obviously too restrictive and not practically interesting. Then what is a reasonable assumption? Here, we want to formulate a more feasible and yet general (but somewhat imperfect) theory, complementing Valiant's learning model, where 'simple' things are easily learnable. In practical learning system designs, when something is not known to be polynomially learnable in Valiant's model, it may be polynomially learnable under our assumptions. Apart from this, our proposed notion of learning will, to some extent, also deal with item (2).

1.1. Reasonable Assumptions

We propose to study Valiant learnability under *all simple distributions*, which properly include *all* recursive and recursively enumerable distributions. Surprisingly, such class of distributions not only is general but also possesses many nice mathematical properties for learning. The semi-computable distributions include *all* distributions we have a name for: uniform distribution, normal distribution, geometric distribution, Poisson distribution, $\cdots$. Under the assumption of learning under arbitrary unknown simple distributions, we will systematically develop a theory of learning for simple concepts that intuitively should be polynomially learnable.

It is an integral part of the proposed approach to also deal with the problem of inability of sampling according to underlying distributions. In real life the samples are sometimes (or often) provided by some mechanical or artificial means or good-willed teachers, rather than provided according to its underlying distribution. Naturally the simpler examples are provided first. The simpler the example, the more likely it is chosen. Think of the situation where you are the teacher who provides examples of a DFA to a learner. What strings do you choose first? With high probability you choose things like $\lambda, 0, 1, 00, 0^{10}$ rather than 11010111000010100110 (obtained by flipping a quarter on a Friday night). In the subconscious mind of the teacher, she tries to help the students. The teacher, with no idea of how the students are programmed, teaches "+" by using example "1+1=2" first rather than "5697+1692=7389". *She uses good examples from her past experiences.* So the sampling distribution and the real distribution may be quite different. The notion of a sampling distribution where 'simple' examples have high probability, and 'complex' examples have low probability, we capture by the *single* 'universal' recursively enumerable distributions (rather, semimeasures) $\mathbf{m}$ and $\mathbf{M}$, or an effective t-timelimited approximation m_t. We conceive of these distributions as being precomputed and stored once, and henceforth accessible from a table**.

Consider another situation where a robot wants to learn but there is nobody around to provide it with examples according to the real distribution. Because it does not know the real distribution, the robot just has to generate its own examples according to its own (computable) distribution and experiment to classify these examples (See [RS]). For example, the robot wants to learn a finite-state automaton concealed in a black box (with resetting mechanism and observable accepting/ rejecting behavior).

1.1.1. Outline

First we deal with discrete sample spaces. We first derive a completeness result: there exists a 'universal simple' distribution such that something is learnable under this *single* distribution iff it is polynomially learnable in Valiant's sense under *all* 'simple' distributions, provided we sample according to the 'universal' distribution. We use this completeness result as a novel tool to obtain new non-trivial learning algorithms for several (old and new) classes of problems, in this model. These classes were not known to be polynomially learnable under Valiant's more general assumptions, some were even NP-complete. For example, the class of DNF's such

** One may speculatively think that this coincides with the following imaginary brain model: A child inherits from her parents a "table" of distribution $\mathbf{m}(x)$. The child "samples" the outside world according to her own "subconscious" mind and learns. Each generation, the table evolves to be more perfect and approximating to $\mathbf{m}(x)$. Our theory then guarantees that the child with the better table has more ability to learn.

that each monomial has Kolmogorov complexity $O(\log n)$, the class of k-reversible DFA of Kolmogorov complexity $O(\log n)$, and the class of k-term DNF, are polynomially learnable under our assumptions. We then turn to continuous sample spaces. While each discrete concept class is learnable in unbounded time, this is not the case for concept classes over continuous sample spaces. Using the 'universal' semimeasure, we show that a concept class is learnable under a simple (semi)measure iff it is learnable under the 'universal' semimeasure. (This result holds both if we sample according to the simple (semi)measure itself, or according to the 'universal' semimeasure.)

2. Preliminaries

Definition 1. (1) Let X be a set. A *concept* is a subset of X. A *concept class* is a set $C \subseteq 2^X$ of concepts. An *example of a concept* $c \in C$ is a pair (x,b) where $b=1$ if $x \in c$ and $b=0$ otherwise. A *sample* is a set of examples.

(2) Let $c \in C$ be the target concept and P be a distribution on X. Given *accuracy parameter* ϵ, and *confidence parameter* δ, a *learning algorithm* A draws a sample S of size $m_A(\epsilon,\delta)$ according to P, and produces a *hypothesis* $h = h_A(S) \in C$.

(3) We say C is *learnable* if for some A in above, for every P and every $c \in C$,

$$P(h \Delta c > \epsilon) \leqslant \delta,$$

where Δ denotes the symmetric difference. In this case we say that C is $(\epsilon,\delta)-$*learnable*, or *pac-learnable (probably approximately correct)*.

(4) C is *polynomially learnable* if A runs in polynomial time (and asks for polynomial number of examples) in $1/\delta$, $1/\epsilon$, and the length of the concept to be learned.

Remark. A different model as used by [V,KLPV] assumes separate distributions over positive and negative examples. These models are basically equivalent. Also see [HLW] for an on-line model.

Definition 2. Let Q be the set of rational numbers, and let S be a countable (i.e., discrete) sample space (like the natural numbers). A function $f: S \to (-\infty, \infty]$ is *recursively enumerable*, or simply *enumerable*, if the set

$$\{(x,r): x \in S, r \in Q, r < f(x)\}$$

is recursively enumerable. (Such functions are also called 'semicomputable'.) A function is called *co-enumerable* if $-f$ is enumerable. (Another, equivalent, definition is: function f is enumerable iff there is a recursive function with rational values $g^n(x)$ nondecreasing in n, with $f(x) = \lim_{n \to \infty} g^n(x)$.) A real function f is *recursive* if it is enumerable and co-enumerable. For example, the Kolmogorov complexity $K(x \mid y)$ (defined below) is co-enumerable. But it is not recursive.

We call a function $w: S \to [0,1]$ a *semimeasure* if

$$\sum_{x \in S} w(x) \leqslant 1. \tag{2.1}$$

We call it a *measure*, or a *probability distribution* over the space S if equality in (2.1) holds.

Definition 3. An enumerable semimeasure μ is called *universal* if it *multiplicatively dominates* every other enumerable semimeasure μ', *i.e.*, $\mu(x) \geqslant c\mu'(x)$ for a fixed constant c independent of x.

Theorem A. [Levin] There is a universal enumerable semimeasure.

Proof. As this proof, and that of Theorem B, to our knowledge are not (easily) accessible elsewhere, we include our versions based on [G] in Appendixes 1 and 2. □

We now fix a universal semimeasure and denote it by $\mathbf{m}(x)$. Any positive function $w(x)$ such that $\sum_x w(x) \leqslant 1$ has to converge to 0. However, $\mathbf{m}(x)$ converges to 0 slower than any positive recursive function which converges to 0. Obviously, this also implies that $\mathbf{m}(x)$ cannot be recursive.

Definition 4. A set of finite strings is a prefix-code if no element is a proper prefix of any other element. We consider a special type of three tape Turing machines with one-way input tape, one-way output tape, and a two-way work tape. The input tape contains an unbounded string of zeros and ones, and the initial segment scanned by the machine by the time it halts is called the *program*. The (self-delimiting) descriptional complexity K_T of x, *relative* to Turing machine T (which by construction halts on programs that form a prefix-code), and binary string y (which is put on the work tape initially), is defined by

$$K_T(x \mid y) = \min\{|p| : p \in \{0,1\}^*, T(p,y) = x\},$$

or ∞ if such p do not exist.

There is a Turing machine U (like a universal Turing machine), such that for each Turing machine T, there is a constant c_T, such that for all x and y, we have $K_U(x \mid y) \leqslant K_T(x \mid y) + c_T$. This definition is invariant up to a constant with respect to different universal Turing machines. Hence we fix a reference universal Turing machine U, and drop subscript U by setting $K(x \mid y) = K_U(x \mid y)$. $K(x)$ is defined as $K(x \mid \epsilon)$. We refer the reader to our survey paper [LV1].

Definition 5. [Solomonoff, Levin] The *universal prior probability* $P_U(x)$ of a finite binary string x is defined as

$$P_U(x) = \sum_{U(p)=x} 2^{-|p|}$$

The next theorem connects the (self-delimiting) Kolmogorov complexity to the universal distribution $\mathbf{m}$.

Theorem B [Levin]. $-\log \mathbf{m}(x) =$

$-\log P_U(x) + O(1) = K(x) + O(1)$.

Proof. See Appendix 2. □

Corollary. The universal prior probability distribution $\mathbf{m}(x)$ multiplicatively dominates any enumerable distribution.

Since the set of programs of U is a prefix-code (no program is a proper prefix of any other program), we have $\sum_{U(p)<\infty} 2^{-|p|} \leq 1$ by Kraft's inequality (see any textbook on information theory or [LV1,LV2]). Therefore, $P_U(x)$ is a semimeasure. Following Solomonoff, we can view P_U as follows. On the input tape of U, one supplies a random coin-flip sequence. By Theorem B, the probability of outputting a string x is $\mathbf{m}(x)$ (up to a constant factor). Obviously this procedure is non-recursive. However, using this procedure dovetailing the computations for all programs p of length $\leq |x|$, and adding $2^{-|p|}$ for each program p of which the computation halts, we can compute $\mathbf{m}(x)$ from below. By Theorem B, it suffices to start only programs p such that $|p| \leq K(x)$ (and $K(x)$, it is known, satisfies $K(x) \leq |x| + 2\log|x| + O(1)$). So we may need to start exponentially many programs in terms of length $|x|$. But, if x is 'simple', that is, $K(x) = O(\log|x|)$, then we only need to start polynomially many computations (namely, for programs p with $|p| \leq K(x)$) to compute $\mathbf{m}(x)$ exactly. We provide a time bounded version of the above discussion in one of the next sections.

We will also need Chernoff's bounds for independent Bernoulli trails which is stated below (See [AV]). Assume that we perform a sequence of n independent Bernoulli trails with probability of success p each time. Let m denote the number of successes. Then, for $\alpha < 1$,

$$P(m - np > \alpha np) < e^{-\frac{1}{2}\alpha^2 np}, \qquad (2.2)$$

$$P(np - m > \alpha np) < e^{-\frac{1}{3}\alpha^2 np}. \qquad (2.3)$$

Definition 6. A distribution $P(x)$ is *simple* if it is (multiplicatively) dominated by a enumerable distribution $Q(x)$. That is, there is a constant c such that for all x,

$$cQ(x) \geq P(x).$$

The first question is how large the class of simple distributions is. It certainly includes all enumerable distributions and hence all distributions in our statistics books. We next show that containment is proper and not vacuous.

Lemma 1. *There is a non-enumerable distribution that is simple.*

Proof. Consider the distribution

$$P(x) = \begin{cases} c/x^2 & \text{if } x \in A \\ 0 & \textit{otherwise} \end{cases}$$

The constant c is determined such that $\sum_x P(x) = 1$. Now if we choose A as a non-r.e. set, $P(x)$ is not enumerable. But $P(x)$ is multiplicatively dominated by the recursive distribution $Q(x) = c'/x^2$ for all x and $c' = \frac{6}{\pi^2}$ is such that $\sum_x Q(x) = 1$. By trivial modification of above, we can also guarantee that $2Q(x) \geq P(x)$ for all x. □

Lemma 2. *There is a distribution which is not simple.*

Proof. We define a probability distribution $f(x)$ which exceeds $x\,\mathbf{m}(x)$ for infinitely many x. Then by Theorem A, $f(x)$ is not simple. Let $l(x)$ be the greatest monotonic lower bound on $\mathbf{m}(x)$, i.e., $l(x) = \inf\{\mathbf{m}(y): y \geq x\}$. ($l(x)$ is approximately $1/(x\log x \log\log x ...)$.) Let $u(x)$ be the least monotonic upper bound on $\mathbf{m}(x)$, i.e., $u(x) = \sup\{\mathbf{m}(y): y \geq x\}$. (It can be shown that $u(x)$ is not recursive - it goes to zero slower than any recursive function.) The desired function $f(x)$ is defined as follows. For each $x = y_2$ of a triple y_1, y_2, y_3 such that $u(y_1) = \mathbf{m}(y_1)$, $l(y_2) = \mathbf{m}(y_2)$, and $u(y_3) = \mathbf{m}(y_3)$, $y_3 - y_1$ minimal, we set $f(x) = u(x)$. For all remaining x, we set $f(x) = 0$. By this definition, for infinitely many x we have $f(x) = u(x)$ while $\mathbf{m}(x) = l(x)$. Therefore, $f(x) > x\,\mathbf{m}(x)$ for infinitely many x. (In fact, the ratio $u(x)/l(x)$, and hence $\sup f(x)/\mathbf{m}(x)$, rises faster than any recursive function). □

Motivation. In a practical situation, if one can dominate the real distribution by a recursive distribution, then the theory we develop in this paper can be used to learn. Take a simple example, if the real life distribution is almost uniform, then it is possible to dominate such a distribution by a uniform distribution which is recursive. It is well-known that many real life distributions obey recursive distributions: the distribution of flying bombs hitting south London during World War II, the number of telephone connections to wrong numbers (during the 1920's), the number of α-particles emitted by a radioactive substance reaching a given portion of space during time t, all obey Poisson distribution $p(k;\lambda) = e^{-\lambda}\frac{\lambda^k}{k!}$ [F]. It is important to notice that when some real distributions can be approximated (hence dominated) by recursive distributions with high probability, it is sufficient for our learning purpose.

3. Distribution-Free Learning when the Distribution is Simple: Discrete Case

In this section all concept classes we deal with are over discrete sample spaces.

Definition 7. The learning algorithm *samples according to* $\mathbf{m}(x)$, if in the learning phase the algorithm draws random samples from $\mathbf{m}(x)$.

We can formalize this in different ways. The following two implementations are equivalent.

(1) The learning algorithm is equipped with an

$\mathbf{m}(x)$ oracle that supplies samples according to $\mathbf{m}(x)$. Intuitively, this is natural in a teacher-student situation, or auto-learning by non-random experiments. In such circumstances samples with low Kolmogorov complexity are drawn with high probability. (Cf. Introduction)

(2) The algorithm has access to an $\mathbf{m}$ table in the form of a division of the real interval [0, 1) into nonintersecting halfopen subintervals I_x such that $\bigcup I_x = [0, 1)$. For each x, the length $|I_x| = \mathbf{m}(x)/\sum_y \mathbf{m}(y)$. Define the *cylinder* Γ_r as the set of all infinite binary strings starting with r. To draw a random example from $\mathbf{m}$, the algorithm uses a sequence $r_1 r_2 \cdots$ of outcomes of fair coin flips until the cylinder Γ_r, $r = r_1 r_2 \cdots r_k$, is contained in some interval I_x. It is easy to see that this procedure of selecting x, using a table $\mathbf{m}$ and random coin flips, is equivalent to drawing a random x according to distribution $\mathbf{m}$. The table $\mathbf{m}$ is of course nonrecursive. However, in certain learning algorithms we consider only examples of fixed length n, which allows us to precompute a timelimited version of $\mathbf{m}$, cf. below.

This model for learning has the following *completeness property*.

Theorem 1. *A concept class C is polynomially learnable under the universal distribution $\mathbf{m}(x)$, iff it is also polynomially learnable under any unknown simple distribution, P, provided the samples are drawn according to $\mathbf{m}(x)$.*

Proof. $P(x)$ is dominated by some enumerable distribution $Q(x)$. $Q(x)$ is in turn dominated by $\mathbf{m}(x)$. Hence, there is a constant $c > 0$ such that for all x,

$$cm(x) \geqslant P(x)$$

Assume C is learnable (in time t) under distribution $\mathbf{m}(x)$. Then one can run the learning algorithms with error parameter ϵ/c in polynomial time. Let *err* be the set of strings that are misclassified by the learned concept. So with probability at least $1 - \delta$

$$\sum_{x \in err} m(x) \leqslant \epsilon/c.$$

Hence

$$\sum_{x \in err} P(x) \leqslant c \sum_{x \in err} \mathbf{m}(x) \leqslant \epsilon.$$

Therefore, if the underlying distribution is $P(x)$ rather than $\mathbf{m}(x)$, we are still guaranteed to 'pac-learn' C (in time t), if sampling according to $\mathbf{m}(x)$. □

In the next sections we show how to exploit this completeness Theorem to obtain new learning algorithms. After all, if we know the sample space has a simple distribution, then we can learn using any learning algorithm for the specific distribution $\mathbf{m}(x)$. The latter distribution has the remarkably convenient property that in a polynomial sample *all* examples of logarithmic complexity occur with probability near one.

The constant c relates to the size of the machine that computes P, that is, if $P(x)$ is enumerated by the ith Turing machine in the standard enumeration, then $c = i^2$. As another remark, obviously, Theorem 1 also holds if we replace $\mathbf{m}$ by any distribution Q that dominates P.

Since $\mathbf{m}$ assigns higher probabilities to simpler strings, one could suspect that after polynomially many examples, *all* simple strings are sampled and the strings that are left unsampled have only very low (inverse polynomial) probability. However, the next theorem shows that this is not the case.

Theorem 2. *Let S be a set of n^c samples drawn according to $\mathbf{m}$. Then*

$$\sum_{x \notin S} m(x) = \Omega\left[\frac{1}{(\log n)^2}\right]. \tag{3.1}$$

Proof. Consider the first n^{c+2} strings. These strings have Kolmogorov complexity at most $(c+2)\log n + 2\log\log n + O(1)$ each. The total probability for these strings, excluding S, is at least

$$\frac{1}{2}(n^{c+2} - n^c)\Omega\left[\frac{1}{n^{c+2}(\log n)^2}\right] = \Omega\left[\frac{1}{(\log n)^2}\right]$$

By using more efficient prefix coding, equation (3.1) can be improved to

$$\sum_{x \notin S} \mathbf{m}(x) = \Omega\left[\frac{1}{\log n \log\log n \log\log\log n \cdots}\right].$$

□

Remark. This says that we cannot just do polynomial sampling and hope to do trivial learning by listing the examples in a table, if error e.g. $\epsilon \leqslant n^{-1}$ is expected.

3.1. Polynomial Time Setting

Now let us consider polynomial time computable distributions. Again, all textbook distributions we know are polynomially computable. Call a distribution *polynomial simple* if it is dominated by a polynomial time computable distribution. In all of the discussion below all Kolmogorov complexity (including the related notion $\mathbf{m}$) can be replaced by its polynomial time bounded version. We will generally use distribution $\mathbf{m}(x)$. But all the results apply equally to the polynomially bounded m_t.

It is convenient to formulate in terms of distribution functions $\mu: \{1,2, \cdots\} \to [0,1]$, where $\mu(x)$ is the probability of all instances not exceeding x. Its density $\mu'(x) = \mu(x) - \mu(x-1)$ is the probability of example x. In the discrete setting all concepts are learnable with unbounded amount of time (by exhaustive sampling) [BI]. We deal with polynomial time learnability.

Remember that $\mathbf{m}(x) = 2^{-K(x)+O(1)}$. One naturally wants to replace this by a time bounded version as follows.

Theorem 3. *If distribution μ is computable in time $t(n)$, then there is a constant c, such that for all x:*

$$-\log\mu'(x) \geqslant -\log c + K_{nt(n)}(x),$$

where $\mu'(x) = \mu(x) - \mu(x-1)$ and $K_f(x)$ is the f-time bounded Kolmogorov complexity of x.

Proof. Let μ be computable by M, and let $c = |M|^2$. Notice that $\sum_x \mu'(x) \leqslant 1$. We wish to show that $K_{nt(n)}(x) \leqslant -\log\mu'(x) + \log c$. Without a polynomial bound, a proof similar to that of the optimality of the Shannon-Fano code would be sufficient. But we have to deal with the time bound here.

We will divide the real interval [0,1] into subintervals such that the code word $p(x)$ for source word x 'occupies' $[\mu(x-1), \mu(x)]$. The *binary interval* determined by the finite binary string r is the half open interval $[0.r,\ 0.r + 2^{-|r|})$ corresponding to the set of reals (cylinder) Γ_r consisting of all reals $0.r....$ The encoding function f assigns to x the code corresponding to a maximum binary interval within $[\mu(x-1), \mu(x))$, of length $\mu'(x)$, and this code has at most $-\log\mu'(x) + 2$ bits.

We have to give polynomially encoding and decoding algorithms. The encoding algorithm is trivial: since μ is computable in time $t(n)$, given source word x, its code word $p(x)$ can be computed from $\mu(x-1)$ and $\mu(x)$ in $O(t(n))$ time. In order to compute p^{-1}, the decoding function, given a code word $p(x)$, we proceed as follows.

Decoding Algorithm.

(1) Set $k := 1$.

(2) Repeatly set $k := 2k$ until $p(x)$ lays in or left of interval $[\mu(k-1), \mu(k)]$. Set $u := k$ and $l := k/2$.

(3) (Binary search) Let $m = (u+l)/2$. If $p(x)$ corresponds to a maximum binary interval in $[\mu(m-1), \mu(m)]$, then return $x = m$. Otherwise set $u := m$ if $p(x)$ lays left of $\mu(m-1)$ and set $l := m$ if $p(x)$ lays right of $\mu(m)$.

This procedure is similar to a binary search, and it takes at most

$$t'(|x|) = \sum_{i=0}^{\log x} t(|\frac{x}{2^i}|) \leqslant O(nt(n))$$

(where $n = |x|$) time to find x if $t(n)$ is a polynomial.

This completes our encoding of x using distribution μ. Notice the constant c comes in because we needed the machine which computes μ to specify each x. Hence

$$K_{nt(n)}(x) \leqslant -\log\mu'(x) + \log c.$$

□

We denote the t-time bounded version of $\mathbf{m}$ by m_t'. That is, $m_t'(x) = m_t(x) - m_t(x-1)$, where

$$m_t(x) = \sum_{y \leqslant x} 2^{-K_t(y)}$$

Lemma 3. *The probability $m_t'(x) = 2^{-K_t(x)}$ can be generated in time polynomial in t, provided $K_t(x) = O(\log|x|)$.*

Proof. Again use a universal Turing machine with one input tape and one output tape. On the input tape it is provided a random coin-flip sequence. The universal machine finds the first initial segment of the coin-tossing sequence which constitutes a program in a prefix-free code. Such a program must be in form of $(n,t(n),p)$. If this is not the case, discard the code. Otherwise simulate the program as follows: Simulate p for $t(n)$ steps. If p stops within $t(n)$ steps, then print the output of p, otherwise print 0^n on the output tape. Thus the probability of generating a string x is approximately $2^{-K_t(|x|)}$. If $K_t(x) = O(\log|x|)$, we can try all programs p of length $|p| \leqslant K_t(x)$, and have the whole process of determining $m_t'(x) = 2^{-K_t(x)}$ run in time polynomial in $t(|x|)$. □

In time polynomial in $t(n)$ we can find all x of length n with $K_t(x) = O(\log n)$, and determine their probabilities $m_t'(x)$. However, for us the following corollary is more important.

Corollary. To compute a table $m'_p(x+1), ..., m'_p(x+2^n)$ takes time $O(p(n)2^n)$, with $p(n)$ is polynomial in n. In other words, we can divide [0, 1) into 2^n half open disjoint intervals I_y with $|I_y| = m'_p(y)/(m_p(x+2^n) - m_p(x))$, such that $\bigcup_y I_y = [0,1)$, $y = x+1, ..., x+2^n$.

Hence, if we want to learn a concept class using examples of fixed size n, sampling according to m'_p, we can *precompute* the interval representation of the table (as in the Corollary) once and for all, and use it to sample according to m_p by means of a sequence of fair coin flips as explained earlier.

3.2. Learning under m(x): log n - DNF

We have established that learning under the universal distribution is important since if one can pac-learn under the (time-bounded) universal distribution, then one can pac-learn, using the same algorithms, under any enumerable distribution by using the (time-bounded) universal distribution and sampling or asking queries according to it. Are there classes of concepts which are not (known to be) learnable under all distributions (in the sense of Valiant) but which are learnable while sampling according to $\mathbf{m}$? We first consider a class for which it is not known whether it is Valiant learnable.

DNF stands for 'disjunctive normal form'. A DNF is any sum $m_1 + m_2 + ... + m_r$ of monomials,

where each monomial m_i is the product of some literals chosen from a universe $x_1, \ldots, x_n$ or their negations $x_1, \ldots, \overline{x_n}$. A k-DNF is a DNF where each monomial consists of at most k literals. Recall, that k-DNF is learnable in Valiant's sense [V]. One is inclined to think that also $(n-k)$-DNF is learnable, or the sum of monomial terms such that every 3rd variable is true, or every 7th element is true, ... like $\sum_i x_1 x_i \ldots x_{\lfloor n/i \rfloor i}$, where i ranges from 1 to $f(n)$ for some sublinear function f. Or more generally, expressed in a DNF form:

$$\sum_{\phi} x_{\phi(1)} x_{\phi(2)} \cdots x_{\phi(n)}, \qquad (3.2)$$

the sum taken over a set of total recursive functions ϕ of cardinality polynomial in n, $\phi(i) \leq n$ for $i = 1, \ldots, n$, should also be learnable. It is not known whether such formulae are Valiant learnable. We show they are learnable in our sense.

Let us write $\log n$-DNF to denote DNF formulae over n variables, where each monomial term is of Kolmogorov complexity $O(\log n)$, and the length of the formula does not exceed a polynomial in n. This is a superset of k-DNF (it contains all formulae of the form (3.2)).

Theorem 4. *$\log n$-DNF is polynomially learnable under* $\mathbf{m}(x)$.

Proof Sketch. Let $f(x_1, \cdots, x_n)$ be a $\log n$-DNF where each term has Kolmogorov complexity $\leq c \log n$. If m is a monomial of f, we write $m \in f$. Sample $n^{c'}$, for $c' > c+1$, examples. With high probability (using the Chernoff formulas (2.2) and (2.3)) all examples of the following form will be drawn.

> For each monomial term m of f, the example vector that satisfies m and has zero values for all variables not in m, denoted by 0_m; the example vector that satisfies m and has one values for all variables not in m, denoted by 1_m.

Now we approximate f by the following learning procedure.

Learning Algorithm

(0) Sample $n^{c'}$ examples according to $\mathbf{m}(x)$. Let *Pos* (*Neg*) be the set of positive (negative) examples sampled.

(1) For each pair of examples in *Pos*, construct a monomial which contains x_i if both vectors have '1' in position i, contains $\overline{x}_i$ if both vectors have '0' in position i, and does not contain variable x_i otherwise.

(2) Among these monomials delete the ones that are inconsistent with negative examples in *Neg*. The remainder forms a set S. Therefore, S contains only monomials which do not imply any negative samples in *Neg*.

(3) Let $E_m = \{x \mid m(x) = 1\}$, that is, E_m is the set of positive examples implied by monomial m.

> Use a greedy set cover algorithm to find a small set of monomials $m \in S$, such that $\bigcup E_m$ covers all positive examples in *Pos*.

We have to prove the correctness of the algorithm.

Claim 1. With high probability, $\{m \mid m \in f\} \subseteq S$.

Proof. Since for each monomial m in f, 1_m and 0_m both have low Kolmogoro complexity, they are sampled with high probability, by standard calculation. >From 1_m and 0_m, one forms m in step (1) of the algorithm. So with high probability, for *each* monomial m of f, we have $m \in S$. By using polynomially more samples, we also have with high probability for *all* $m \in f$, $m \in S$. □

Of course, many other monomials may also be in S. Finding all of the original monomials of f precisely is NP-hard. For the purpose of learning it is sufficient to approximate f. We use the following result due to Johnson [J] and Lovasz [Lo],

Claim 2. Given sets $A_1, \cdots, A_n$, such that $\cup_{i=1}^n A_i = A = \{1, \cdots, m\}$. If there exist k sets $A_{i_1}, \cdots, A_{i_k}$ such that $A = \cup_{j=1}^k A_{i_j}$, then it is possible to find in polynomial time $l = O(k \log m)$ sets $A_{i_1}, \cdots, A_{i_l}$ such that $A = \cup_{j=1}^l A_{i_j}$. □

Let f have k monomials. These k monomials cover the positive examples in the sense that $Pos \subseteq \cup_{m \in f} E_m$. By Claim 2, we can use about $O(kn)$ monomials to approximate f and cover positive examples in *Pos* in polynomial time. Then Occam's Razor theorem [BEHW] implies that our algorithm learns $\log n$-DNF in the sense of Definition 1. □

Remark. Notice that we draw examples from $\{0,1\}^n$ according to $\mathbf{m}(x)$. There are 2^n such vectors but there are 3^n monomials of n variables. Hence one cannot trivially encode each monomial term into binary vectors of length n and get all the low complexity terms by sampling. An interesting open question: is $\log n$-decision list polynomially learnable under $\mathbf{m}(x)$? A $\log n$-decision list is a decision list of Rivest [R] with each term having Kolmogorov complexity $O(\log n)$.

3.3. Learning under m(x): Simple DNF

Using the idea of the previous section, we can also learn a more general class of DNF's where each term may have very high Kolmogorov complexity. Let us define a DNF formula f to be *simple* if, for each term m of f, there is vector v_m that satisfies m but satisfies no other monomials of f and $K(v_m) = O(\log n)$. Obviously, simple DNF's can contain many high Kolmogorov complexity terms. The learning algorithm for the class of simple DNF's goes as follows. (Since the basic ideas used are similar to those in the previous section, we skip the details.)

Learning Algorithm.

(0) First we sample enough (polynomially many) examples.

(1) For each positive example, construct the corresponding monomial of size n.

(2) For each monomial m constructed in step (1), mark variable x_i in m if there is a negative example that would satisfy m if the ith bit is complemented. In m delete the unmarked variables. Remove those monomials that are satisfied by some negative examples.

(3) Use the set cover algorithm to choose a small set of monomials that cover all the positive examples (as in the proof of Theorem 4).

For each monomial m in f, there is a vector v_m that satisfies only m and no other monomials in f. Hence if one flips a bit in v_m, that corresponds to a variable in m, it becomes a negative example of Kolmogorov complexity $O(\log n)$. From this v_m and corresponding negative examples (which will all be sampled with high probability), one forms m. There will also be many other monomials. But since all the true monomials of f are in the set, we can cover all positive examples using about $|f|\log n$ monomials in polynomial time. Hence using Occam's Razor theorem, we have learned in the sense of Definition 1.

3.4. Learning Under m(x): Simple Reversible Languages

A deterministic finite automaton (DFA) $\mathbf{A} = (Q, q_0, F, A, \delta)$ consists of a set of states Q, a finite nonempty input alphabet A, an initial state q_0 and a set of final states $F \subseteq Q$, and a transition function $\delta: Q \times I \to Q$. $\mathbf{A}$ is *0-reversible* if it has only one final state ($|F| = 1$), and its reversal $\mathbf{A}^R$ is deterministic. ($\mathbf{A}^R$ is obtained from $\mathbf{A}$ by reversing each transition in $\mathbf{A}$ and exchange the initial and the final state of $\mathbf{A}$.) An alternate definition would be that $\mathbf{A}$ is *0-reversible* if it is deterministic with one initial state and one final state and for no q_1 and q_2 is $\delta(q_1,a)=\delta(q_2,a)$ for some $a \in A$. A language L is 0-reversible if it is accepted by a 0-reversible DFA.

Many languages/DFA's are 0-reversible and of low Kolmogorov complexity. Examples are, for fixed n, the language $L_1 =$ set of strings of length at least n and containing an even number of zeroes, and the language $L_2 = \{0^k 1^j \mid k,j \geq n\}$.

Recall that a nondeterministic finite automaton (NFA) is like a DFA with q_0 replaced by $I \subseteq Q$, and $\delta: Q \times A \to 2^Q$. We generalize 0-reversibility as follows. A *k-reversible* DFA is a DFA $\mathbf{A}$ such that in (the possibly nondeterministic) $\mathbf{A}^R$, if two distinct states q_1, q_2 are initial states or $q_1, q_2 \in \delta(q_3, a)$ for $a \in A$, then no string u of length k satisfies both $\delta(q_1,u) \neq \emptyset$ and $\delta(q_2,u) \neq \emptyset$. (This guaranties that any nondeterministic choice in the operation of $\mathbf{A}^R$ can be resolved by looking ahead k symbols past the current one.) A language is k-reversible if it is accepted by a k-reversible automaton.

For each fixed n, $L_3 = \{0^k 1^m \mid k \geq n, m \geq 1\}$ and $L_4 = \{0^m 1^k \mid k \geq n, m \geq 1\}$ are 1-reversible and have $O(\log n)$ Kolmogorov complexity. (The construction of the corresponding automata is left as an exercise.) We say a *path* from the initial state to a final state is *simple* if it has Kolmogorov complexity $O(\log n)$. A k-reversible DFA $\mathbf{A}$ is *simple* if each state of $\mathbf{A}$ lies on a simple path.

Example. We give some examples of simple k-reversible DFA's. Firstly, the set of k-reversible DFA's of Kolmogorov complexity $O(\log n)$ are simple. To see this, consider $\mathbf{A}$ such that $K(\mathbf{A}) = c \log |\mathbf{A}|$. We show that every state of $\mathbf{A}$ is on a simple path of Kolmogorov complexity at most $(c+1)\log |\mathbf{A}|$. Without loss of generality, assume that every state of $\mathbf{A}$ is reachable from the initial state. (If this is not the case, we can just delete those obsolete states from $\mathbf{A}$.) We have assumed that $\mathbf{A}$ can be specified in $c \log n$ bits. Fix an enumeration of the paths from the initial state to final states of $\mathbf{A}$ such that each path contains at least one more new state. There are at most $|\mathbf{A}|$ such paths since each path must contain at least one new state which is not contained in the previous paths. Obviously, each such path can be specified using $\mathbf{A}$, that is, $c \log n$ bits, and the index of the path in $\log n$ bits. Hence the Kolmogorov complexity of each such path is at most $c \log n + \log n$. Every state is on at least one of these paths by construction.

Notice that if $K(\mathbf{A}) = O(\log n)$, it is still possible that $\mathbf{A}$ may have very random paths. For example, the automaton which accepts all strings of length n has Kolmogorov complexity $O(\log n)$, but it actually contains a path for every string of length n. In particular, it contains a path of Kolmogorov complexity n. On the other hand, one can construct (left to the reader) a simple 0-reversible DFA which has Kolmogorov complexity much larger than $\log n$ (like $\Omega(\log^2 n)$).

In the general Valiant distribution free setting, it is not known whether the class of 0-reversible languages is learnable. Angluin [A1] shows that the set of k-reversible languages can be identified in the *limit* in the Gold paradigm. In general a DFA is learnable by membership queries and equivalence queries [A1] in polynomially.

Theorem 5. *The class of simple k-reversible automata is polynomially learnable under* $\mathbf{m}$.

Proof. We first show how to learn a simple 0-reversible DFA under the universal distribution.

Claim 1. The class of 0-reversible DFA of Kolmogorov complexity $O(\log n)$ is learnable.

Proof. The algorithm uses the ideas in [A1], [BF], [M].

Learning Algorithm.

(1) Randomly sample n^{c+2} positive examples. Construct the trivial tree DFA from these examples.

(2) Merge all the final states in above tree.

(3) *repeat*

if there are states $p,q \in Q$ such that on input $a \in A$, p,q lead to the same state, then merge p and q

until no more merges.

We prove that this algorithm correctly infers the underlying DFA **A** with high probability. Each positive example represents a path from q_0 to q_f, where some states may be repeated because of loops in **A**.

By the use of Chernoff bounds (2.2) and (2.3), with high probability, all simple paths are sampled.

Claim 1.1. Given all simple paths of **A**, **A** can be inferred from the above algorithm.

Proof. All states of **A** are presented at least once in the tree constructed above. It is up to the algorithm to merge them correctly. Now between any two simple paths, P_1 and P_2, if there is a transition from a state a on P_1 to a state b on P_2, then the path from q_0 to a via P_1 then to b then to q_f via P_2 is also simple, hence also given, hence the above merging process will add a transition from a to b correctly. Since this applies to all transitions, eventually **A** will be correctly inferred. Then the non-simple strings, which are also sampled since they may have higher than $1/(\log n)^2$ probability in total, will also fit into the structure. Notice that all above merges do not introduce mistakes since we are dealing with 0-reversible DFA's. □

Claim 2. For each k, the class of simple k-reversible DFA is learnable.

Proof. A simple generalization of the algorithm and the proof in Claim 1 shows that simple k-reversible languages are polynomially learnable under **m**, for each fixed k. □

We show how to learn the class of simple k-reversible languages for all k. The algorithm is given as follows:

for k:=1 to ∞ do

Apply the algorithm for learning k-reversible language to learn a k-reversible DFA;

Draw a polynomial set of examples to test the inferred automaton against;

if the DFA learned is consistent with $(1-\epsilon)$ fraction of the test set

then output this DFA
else continue with the next k value;

The correctness and complexity analysis are standard. (Note that $k < n$, where n is the number of states of the inferred DFA.) Notice that it is not necessary that the correct k-reversible DFA is learned, but by Occam's Razor lemma [BEHW] we have achieved our purpose of learning. Analysis is standard and omitted. □

Remark. Claim 1 in the proof of Theorem 5 explains several successful early experiments by Feldman [F] and Miclet [M]. The examples were supplied by humans, hence likely to be simple (and therefore with high probability according to **m**), and the 0-reversible automata they wanted to infer were all simple.

3.5. Learning under m(x): Previously NP-complete Cases

The previous two subsections provided several classes that are polynomially learnable under the universal distribution, and hence in our sense under all simple distributions, and which are not known to be polynomially learnable in the general Valiant model. The purpose of this subsection is to demonstrate a class that was shown to be not polynomially learnable in Valiant's sense, unless P = NP, but which is polynomially learnable under $\mathbf{m}(x)$.

We have defined DNF before. A monomial in a DNF is *monotone* if no variable in it is negated. A k-term DNF is a DNF consisting of at most k monomials. In [PV] it was shown that learning a monotone k-term DNF by k-term (or $2k$-term) DNF is NP-complete (See also [KLPV]).

Theorem 6. *Monotone k-term DNF is polynomially learnable by monotone k-term DNF under* **m**.

Proof. Assume we are learning a monotone k-term DNF $f(x_1, \cdots, x_n) = m_1 + \cdots + m_k$, where m_i's are the k monotone monomials (terms) of f.

Learning Algorithm.

0. Draw a sample of $n^{k'}$ examples, $k' > k+1$. Set DNF $g := \emptyset$. (g is the DNF we will eventually output as approximation of f.)
1. Pick a positive example $a = (a_1, \cdots, a_n)$. Form a monotone term m such that m includes x_i if $a_i = 1$.
2. *for* each positive example $a = (a_1, ..., a_n)$ *do*: if $a_i = 0$ and deleting x_i from m violates no negative examples, delete x_i from m.
3. Remove from the sample all positive examples which are implied by m. Set $g \leftarrow g + m$. If there are still positive examples left, then go to step 1, else halt and return g.

We show that the algorithm is correct. Let us write $m_i \subseteq m$ for two monotone monomials if all the variables that appear in m_i also appear in m. At step 1, the monomial m obviously implies no negative examples, since for some monomial m_i of f we must have $m_i \subseteq m$. Step 2 of the algorithm keeps deleting variables from m. If at any time for no

monomial $m_i \in f$ holds $m_i \subseteq m$, then there exists a negative example that contains at most k 0's such that it satisfies m but no m_i of f. This negative example is of Kolmogorov complexity at most $k \log n$, hence by the Chernoff formulae (2.2) and (2.3), with high probability it is contained in the sample. Hence at step 2, with high probability, there will be an m_i such that $m_i \subseteq m$. Hence we eventually find a correct m_i (precisely) with high probability. Then at step 3, we remove the positive examples implied by this m_i and continue on to find another term of f. The algorithm will eventually output $g = f$ with high probability by standard calculations. □

Remark. Notice that this is not an approximation algorithm like the ones in the previous sections. This algorithm outputs the precise monotone formula with high probability.

4. Distribution-free learning when the distribution is simple: Continuous Case

We consider continuous sample spaces, e.g., $S = \{0, 1\}^\infty$ consisting of all one-way infinite binary strings. A *semimeasure* (or *distribution*) μ on S satisfies (with ϵ the empty word and $x \in \{0, 1\}^*$):

$$\mu(\epsilon) \leqslant 1, \qquad \text{(i)}$$

$$\mu(x) \geqslant \mu(x0) + \mu(x1). \qquad \text{(ii)}$$

The meaning* of $\mu(x)$ is the combined probability (measure) of the set of elements, or *cylinder*, $\Gamma_x \subseteq S$ defined as $\Gamma_x = \{xy : y \in S\}$. A semimeasure is a *measure* if equality holds in (i) and (ii). For example, consider the measure $\lambda(x) = 2^{-|x|}$. That is, the measure of the cylinder Γ_x, of all elements of S which start with x, is $\lambda(x)$. This defines the Lebesgue measure, or uniform measure, on interval [0,1].

While for discrete sample spaces all concept classes are Valiant learnable (although not all are polynomially learnable), this is not the case for continuous sample spaces. We define the notion of 'simple' semimeasure and that of universal enumerable semimeasure, over a continuous sample space, and show that all concept classes are learnable over each simple semimeasure D iff they are learnable under the universal semimeasure. In contrast with the discrete case w.r.t. polynomial learning, here we do not need to require that the learning algorithm samples according to the universal measure. Due to the space limitation, we can only give a rough idea of what is going on in this section.

A *monotonic* machine M is a three tape machine similar to the three tape machine we defined before, but now for all finite p, q, we have $M(pq) = M(p)r$ for some r. Instead of universal distribution $\mathbf{m}$, consider here its continuous version $\mathbf{M}(x)$ which is the probability for the set $\Gamma_x = \{xy \mid y \in \{0,1\}^\infty \cup \{0,1\}^*\}$ under the reference universal monotonic machine if the input is provided by random coin flips. Each enumerable semimeasure over Ω can be defined this way w.r.t. an appropriate monotonic machine.

A semimeasure D over Ω is *simple* if it is dominated by a enumerable semimeasure E, i..e., if there is a $d > 0$ such that $E(x) > d\,D(x)$ for all x. (As above, $E(x)$ and $D(x)$ are the measures of Γ_x.)

Theorem C [Levin]. $\mathbf{M}$ *is a maximal enumerable semimeasure, i.e., for each enumerable semimeasure D over this domain there is a constant $d > 0$ such that $\mathbf{M}(x) > d\,D(x)$. (For details, see [ZL].)*

Theorem 7. *A concept class C is learnable under $\mathbf{M}(x)$ iff it is learnable under each simple semimeasure.*

Proof Idea. The "if" part is trivial. We only need to prove the "only if" part. We use some definitions and results from [BI]. According to [BI] C_ϵ is an $\epsilon-$*cover* of C (w.r.t. distribution D) if for every $c \in C$ there is a $\hat{c} \in C_\epsilon$ which is ϵ-close to c (*i.e.* $D(c \Delta \hat{c}) < \epsilon$). A concept class C is finitely *coverable* if for every $\epsilon > 0$ there is a finite ϵ-cover C_ϵ of C.

Lemma 4 [BI] *C is finitely coverable w.r.t D* iff C is learnable w.r.t. D.

Proof. We briefly give the main idea of the proof of Benedek and Itai.

Only If Part. Assume that C is finitely coverable under D. We show C is learnable under D. This is done by encoding the finite cover set C' of C into the learning algorithm and choosing the concept from C' which make the least amount of error with respect to the drawn data in the learning phase. By standard application of Chernoff bound this algorithm learns with sufficient data.

If Part. Assume that f learns C under D using sample of size l with error $< \epsilon$ with probability $> 1 - \delta$. We show C is finitely coverable under D. Let $n = n(D, C, \epsilon)$ be the cardinality of the smallest 2ϵ-cover of C under D (n may be infinity).

Choose a set $C_{2\epsilon} \subseteq C$ of n pairwise 2ϵ-far concepts. This is possible because the smallest 2ϵ-cover has size n, and $C_{2\epsilon}$ can be constructed by each time choosing a concept that is 2ϵ-far to other concepts in $C_{2\epsilon}$ already until $|C_{2\epsilon}| = n$ (or keep going if $n = \infty$).

Let $\mathbf{x} = (x_1, \cdots, x_l)$ and $\mathbf{L} = (L_1, \cdots, L_l) \in \{0,1\}^l$. Then $(\mathbf{x}, \mathbf{L})$ is the sample of size l, where L_i is the label for data x_i. If $x_i \in c$ iff $L_i = 1$, then we denote this $\mathbf{L}$ by $\mathbf{L}_c$. For $c \in C$, let

$$g_f(c, \mathbf{x}, \mathbf{L}, \epsilon) = \begin{cases} 1 & \text{if } D(f(\mathbf{x}, \mathbf{L}) \Delta c) < \epsilon \\ 0 & \text{otherwise} \end{cases}$$

---

* Note that this definition is different from our definitions for discrete sample space N. Obviously, $\mathbf{m}$ does not satisfy (ii) if we interpret the arguments as natural numbers. The relation is somewhat more sophisticated. Here we only point out that here we mean by $\mu(x)$ the measure of a set of elements (starting with x), while in the discrete case we intended the probability $P(x)$ of the single element x.

Consider the sum

$$S = \sum_{c \in C_{2\epsilon}} \int_{\mathbf{x}} g_f(c, \mathbf{x}, \mathbf{L}_c, \epsilon) dD. \qquad (*)$$

By hypothesis, $Pr_D(g_f(c, \mathbf{x}, \mathbf{L}_c, \epsilon) = 1) > 1 - \delta$ for randomly drawn sample $(\mathbf{x}, \mathbf{L}_c)$ from D, we obtain $S > (1-\delta)n$. Also we have

$$S \leq \sum_{c \in C_{2\epsilon}} \int_{\mathbf{x}} \sum_{\mathbf{L} \in \{0,1\}^l} g_f(c, \mathbf{x}, \mathbf{L}, \epsilon)\, dD$$

$$= \int_{\mathbf{x}} \sum_{\mathbf{L} \in \{0,1\}^l} \sum_{c \in C_{2\epsilon}} g_f(c, \mathbf{x}, \mathbf{L}, \epsilon)\, dD.$$

Since concepts in $C_{2\epsilon}$ are 2ϵ-far, for every $(\mathbf{x}, \mathbf{L})$ there exists at most one $c \in C_{2\epsilon}$ such that $g_f(c, \mathbf{x}, \mathbf{L}, \epsilon) = 1$. Thus

$$(1-\delta)n \leq S \leq \int_{\mathbf{x}} \Big(\sum_{\mathbf{L} \in \{0,1\}^l} 1 \Big) dD = \int_{\mathbf{x}} 2^l\, dD = 2^l.$$

Hence $l \geq \log((1-\delta)n)$. When $n = \infty$, this implies that f has to take an infinite sample. Hence finitely learnable means finitely coverable. □

A similar proof works also for $\mathbf{M}(x)$. So if one can learn C under $\mathbf{M}(x)$, then one can finitely cover C w.r.t. $\mathbf{M}(x)$.

Let D be a simple distribution, and $d > 0$ such that $\mathbf{M}(x) \geq d\, D(x)$ for all x. Then, any finite ϵd-cover of C w.r.t. $\mathbf{M}(x)$ is also a finite ϵ-cover w.r.t. D. Using Lemma 4 again, it follows that C is learnable w.r.t. D. □

Remark. Note that this is a strong statement since we are saying that if one can learn under $\mathbf{M}(x)$, then one can also learn under any simple distribution (semimeasure) D, while sampling according to D.

Obviously, by the proof, if a concept class C is finitely coverable with respect to $\mathbf{M}(x)$, then it is also finitely coverable and hence learnable under any simple distribution $D(x)$.

Is anything that is learnable under all simple distributions also learnable under all distributions?

Theorem 8. *There is a concept class that is learnable under all simple distributions but not learnable under all distributions.*

Proof. Let the concept class C be the class of all finite sets of cylinders Γ_x such that each such set containing at most one cylinder Γ_x for each length $|x|$. Such class has infinite VC-dimension hence not learnable under arbitrary distribution [BEHW]. (To see this: For any d consider a set S of cardinality d. For every subset S' of S we can find a concept $c \in C$ of d cylinders such that $c \cap S = S'$.) We now show that C is finitely coverable under $\mathbf{M}(x)$. There are only finitely many cylinders Γ_x such that $M(\Gamma_x) > \epsilon/2$ for any ϵ, we denote such set of cylinders by S_ϵ for a given ϵ. Since each concept in C contains at most one cylinder Γ_x for each i such that $|x| = i$, for a given ϵ, we can define an ϵ-cover of C as follows,

$$C_\epsilon = \{c' \mid \textit{for some } c \in C,\ c' = c \cap S_\epsilon\}.$$

It can be easily verified that C_ϵ is finite and ϵ-covers C. Therefore C is finitely coverable and by Lemma 4 C is learnable. □

5. Ongoing research

It seems likely, that many simple concepts previously polynomially un-learnable become polynomially learnable in our model We have given evidence for this by several examples. Is $\log n$ decision list - in analogy with $\log n$ DNF - polynomially learnable in our model?. The connection between our approach of sampling under **m** and learning via queries is obvious, but has not been treated here.

6. Average Case Complexity

The general ideas developed below show that the average complexity of any algorithm whatsoever under the universal distribution is of the same order of magnitude as the worst-case complexity. This holds both for time complexity and for space complexity. To focus our discussion, we use as illustrations the particular case of sorting algorithms, and the general case of the average case complexity of NP-complete problems. Technically the proofs are not complicated once the used concepts are understood.

6.1. Two Examples: Sorting and NP-Completeness

It is well-known, that for some sorting algorithms the average case analysis under some distributions on the inputs gives a different running time than the worst-case running time. For instance, using Quicksort on a list of n items to be sorted gives under the Uniform Distribution on the inputs an average running time of $O(n \log n)$ while the worst-case running time is $\Omega(n^2)$. The worst-case running time of Quicksort is typically reached if the list is already sorted or almost sorted, that is, exactly in cases where we actually should not have to do much work at all. Since in practice the lists to be sorted occurring in computer computations are very likely to be sorted or almost sorted, programmers implementing systems involving sorting algorithms tend to resort to fast sorting algorithms of which the provable average run-time is of equal order of magnitude as the worst-case run-time, even though this average running time can only be proved to be $O(n \log^2 n)$ under the Uniform Distribution as in the case of Shellsort.

In the case of NP-complete problems the question arises whether there are algorithms that solve instances fast "on the average", even while there is little hope to solve them fast in the worst-case. This depends on the particular NP-complete problem to be solved and the distribution of the instances. Obviously, some combinations are easy on the average, and some combinations are hard on the average, by tailoring the distribution to the ease or

hardness of the individual instances of the problem. This raises the question of a significant definition of a "hard on the average" problem. Levin [Le] has shown that for the Tiling problem with uniform distribution of instances there is no polynomial on the average algorithm, unless every NP-complete problem with every simple probability distribution has it. Here it is shown that under the Universal Distribution *all* NP-complete problems are hard to compute on the average unless P = NP (this follows from Theorem 9). Since for each probability distribution $P(x)$ we have that $\mathbf{m}(x) < P(x)/k$ with P-probability at least $1 - 1/k$, and for each enumerable probability distribution $P(x)$ there is a constant c such that $P(x) < c\mathbf{m}(x)$ for all x, the same type of reasoning can show that the average time complexity is near the worst-case time complexity with large P-probability provided P is enumerable (in the full paper). This supports the experience that practical algorithms implementing solutions to many NP-complete problems often suffer from an exponentially exploding running time on most instances that occur, even though this would not be predicted by the theoretical analysis assuming, say, the Uniform Distribution.

6.2. Average Case Complexity under the universal distribution

Definition. Let $t(x)$ is the running time of algorithm A on problem instance x. Define the *worst-case time complexity* of A as $T(n) = \max\{t(x): l(x) = n\}$. (So this is independent of the probability distribution of the problem instances.) Define the *average time complexity* of A with respect to a (semi)measure $\mu(x)$ on the problem instances as

$$T^{\mu}_{average}(n) = \frac{\sum_{l(x)=n} \mu(x)\, t(x)}{\sum_{l(x)=n} \mu(x)}.$$

Example (Quicksort). Let us compare the average time complexity for Quicksort under the Uniform Distribution $P(x)$ and the one under the Universal distribution $\mathbf{m}(x)$. We need to encode lists as integers in some standard way.

For Quicksort, with $P(x)$ the Uniform Distribution on the inputs, $T^{P}_{average}(n) = \Theta(n \log n)$. We may expect that $T^{\mathbf{m}}_{average}(n) = \Omega(n \log n)$. But Theorem 9 will tell us much more, namely, $T^{\mathbf{m}}_{average}(n) = \Omega(n^2)$! Let us give some intuition why this is the case. The worst-case complexity $T(n) = \Omega(n^2)$ holds obviously under all distributions. With the low average time-complexity under the Uniform Distribution, there can only be $o((\log n)2^n/n)$ strings x of length n with $t(x) = \Omega(n^2)$. Therefore, given n, each such string can be described by its sequence number in this small set, and hence for each such x we find $K(x \mid n) \leq n - \log n + 3 \log\log n$. (Since n is known, we can find each $n - k$ by coding k self-delimiting in $2\log k$ bits. The inequality follows by setting $k = \log n - \log\log n$.) Therefore, no really random x's, with $K(x \mid n) \geq n$, can achieve the worst-case run time $\Omega(n^2)$. Only strings x which are non-random, with $K(x \mid n) < n$, among which are the sorted or almost sorted lists, and lists exhibiting other regularities, can have $\Omega(n^2)$ running time. Such lists x have relatively low Kolmogoro complexity $K(x)$ since they are regular (can be shortly described), and therefore $\mathbf{m}(x) = 2^{-K(x)}$ is very high. Therefore, the contribution of these strings to the average running time is weighted very heavily. This intuition can be made precise in a much more general form.

Theorem 9. *Let A be any algorithm whatsoever, provided the running time $t(x)$ is recursive. Let the inputs to A be distributed according to the universal distribution $\mathbf{m}(x)$. Then the average case time complexity is of the same order of magnitude as the corresponding worst-case time complexity.*

Proof. We define a probability distribution $P(x)$ on the inputs that assigns high probability to the inputs for which the worst-case complexity is reached, and zero probability for other cases.

Let A be the sorting algorithm involved. Let $T(n)$ be the worst-case time complexity of A. Clearly, $T(n)$ is recursive (for instance by running A on all x's of length n). Define the probability distribution $P(x)$ by:

1 For each $n = 1, 2, \ldots$, define $a_n := \sum_{l(x)=n} \mathbf{m}(x)$;

2 if $l(x) = n$ and x is lexicographically least with $t(x) = T(n)$, then $P(x) := a_n$, else $P(x) := 0$.

It is easy to see that a_n is enumerable since $\mathbf{m}(x)$ is enumerable. Therefore, $P(x)$ is enumerable. We have defined $P(x)$ such that $\sum_x P(x) \leq \sum_x \mathbf{m}(x) = 1$, and therefore $P(x)$ is a semimeasure. Since $\mathbf{m}(x)$ dominates all enumerable semimeasures multiplicatively, for all x we have

$$P(x) \leq c_P\, \mathbf{m}(x),$$

for a fixed positive constant c_P independent of x (but depending on the index of P in the effective enumeration $\mu_1, \mu_2, \ldots$ of enumerable semimeasures). Relation between $P(x)/c$ and $\mathbf{m}(x)$. The average case time complexity $T^{\mathbf{m}}_{average}(n)$ with respect to the $\mathbf{m}(x)$ distribution on the inputs is:

$$\begin{aligned} T^{\mathbf{m}}_{average}(n) &= \frac{\sum_{l(x)=n} \mathbf{m}(x)\, t(x)}{\sum_{l(x)=n} \mathbf{m}(x)} \\ &\geq \frac{1}{c_P} \sum_{l(x)=n} \frac{P(x)}{\sum_{l(x)=n} \mathbf{m}(x)}\, T(n) \\ &\geq \frac{1}{c_P} \sum_{l(x)=n} \alpha \frac{P(x)}{\sum_{l(x)=n} P(x)}\, T(n) \\ &\geq \frac{\alpha}{c_P}\, T(n), \end{aligned}$$

where

$$\alpha = \frac{\sum_{l(x)=n} P(x)}{\sum_{l(x)=n} \mathbf{m}(x)} = 1.$$

□

If P is μ_k in the standard effective enumeration $\mu_1, \mu_2, \ldots$ of enumerable semimeasures, then we can set $c_P = k^2$. This gives a interpretation to the constant of proportionality between the average complexity and the worst-case complexity under the Uniform Distribution on the inputs: if the algorithm to approximate $P(x)$ from below is the kth algorithm in the standard effective enumeration of all algorithms, then:

$$T^P_{average}(n) \geq k^{-2}\, T(n).$$

Hence we must code this algorithm as compact as possible to get the most significant lower bound. That is, the ease with which we can describe (algorithmically) the strings which produce a worst case running time determines the closeness of the average time complexity to the worst-case time complexity.

Corollary. The analogue of Theorem 9 for *space* complexity holds by about the same proof.

Corollary. For each NP-complete problem, if the problem instances are distributed according to **m**, then the average running time of any algorithm that solves it is superpolynomial unless P = NP. (Related considerations occur in [BCGL].)

Acknowledgements.

Benny Chor, Gloria Kissin and John Tromp commented on the manuscript. The latter suggested the need for Lemma 2. Chor pointed out related results in [BCGL]. The incentive to investigate average case complexity under the universal distribution was supplied by a question of Mike O'Donnel during a lecture given by the second author.

References

[AV] D. Angluin and L. Valiant, Fast probabilistic algorithms for Hamiltonian circuits and matchings, JCSS, 18(1979), pp. 155-193.

[A1] D. Angluin. Learning Regular Sets From Queries and Counter-examples, Yale TR-464, 1986.

[AL] D. Angluin and P.D. Laird, Identifying k-CNF Formulas From Noisy Examples, Yale TR-478, 1986.

[A2] D. Angluin, On the Complexity of Minimum Inference of Regular Sets, Inform. and Control, 39(1978), pp. 337-350.

[A3] D. Angluin, Inference of Reversible Languages, JACM, 29(1982), pp. 741-765,

[AS] D. Angluin and C. Smith, Inductive Inference: Theory and Methods, Comput. Surveys, 15(1983), pp. 237-269.

[BCGL] S. Ben-David, B. Chor, O. Goldreich, M. Luby, On the theory of average case complexity, Proc. 21th STOC, 1989, pp. 204-216.

[BI] G. Benedek and A. Itai, Learnability by fixed distributions, Proc. 1st ACM COLT, 1988, pp. 80-90.

[BR] P. Berman and R. Roos, Learning one-counter language in polynomial time, Proc. 28th FOCS, 1987, pp. 61-77.

[BF] A. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior, IEEE Trans. Comput. C-21(1972), pp. 592-597.

[BEHW] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, Classifying Learnable Geometric Concepts With the Vapnik-Chervonenkis Dimension, Proc. 18th STOC, 1986, pp. 273-282.

[BEHW1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, Occam's Razor, Information Processing Letters, 24(1987), pp. 377-380.

[F] W. Feller, An introduction to probability theory and its applications, Wiley, 1950.

[G] Gács, P., Lecture notes on descriptional complexity and randomness, Manuscript, Boston University, Boston, Mass., October 1987 (Unpublished).

[G1] Gold, E.M., Language Identification in the Limit. Information and Control, 10(1967), pp. 447-474.

[G2] Gold, E.M., Complexity of Automaton Identification from Given Data, Information and Control, 37(1978), pp. 302-320.

[H] D. Haussler, Quantifying the Inductive Bias in Concept Learning, Technical Report, U.C. Santa Cruz, UCSC-CRL-86-25.

[HLW] D. Haussler, N. Littlestone, and M. Warmuth, Expected Mistake Bounds for On-Line Learning Algorithms, Proc. 29th FOCS, 1988, pp. 100-109.

[I] S.Y. Itoga, A new heuristic for inferring regular grammars. IEEE Trans. PAMI-3(1981), pp. 191-197.

[J] D. Johnson, Approximation algorithms for combinatorial problems, JCSS, 9(1974), pp. 256-276.

[KL] M. Kearns and M. Li, Learning in the Presence of Malicious Errors, Proc. 20th STOC, 1988, pp. 267-280.

[KLPV] M. Kearns, M. Li, L. Pitt, and L.G. Valiant, On the Learnability of Boolean Formulae. 19th ACM Symposium on Theory of Computing, 1987, pp. 285-295.

[KV] M. Kearns and L. Valiant, Learning Boolean formulae or finite automata is as hard as factoring, Proc. 21th STOC, 1989, pp. 433-444.

[LV1] M. Li and P. Vitanyi, Two decades of Applied Kolmogorov complexity, 3rd IEEE Structure in Complexity Theory conference, 1988, pp. 80-101.

[LV2] M. Li and P. Vitanyi, Inductive reasoning and Kolmogorov complexity, 4th IEEE Structure in Complexity Theory conference, 1989, pp. 165-185.

[le] L.A. Levin, Average case complete problems, SIAM J. Comp., 15(1986), pp. 285,286.

[L] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear Threshold Algorithm, 28th IEEE Symposium on the Foundations of Computer Science, 1987, pp. 68-77.

[Lo] L. Lovasz, On the ratio of optimal integral and fractional covers, Discrete Math, 13(1975), pp. 383-390.

[M] L. Miclet, Regular inferences with a tail-clustering method. IEEE Trans. Syst. Man Cybern. 10(1980), pp. 737-743.

[N] B.K. Natarajan, On Learning Boolean Functions, 19th STOC, 1987, pp. 296-304.

[PV] L. Pitt and L.G. Valiant, Computational Limitations on Learning From Examples, JACM 35(1989), pp. 965-984.

[PW] L. Pitt and M. Warmuth, Reductions among Prediction Problems: On the difficulty of Predicting automata, Proc. IEEE 3rd Structure in Complexity Conference, 1988, pp. 60-69.

[PW] L. Pitt and M. Warmuth, The minimum consistent DFA problem cannot be approximated within any polynomial, Proc. 21st STOC 1989, pp. 421-432.

[R] R. Rivest, Learning Decision-Lists, Machine Learning, 2(1987), pp. 229-246.

[RS] R. Rivest and R. Schapire, Diversity-based inference of finite automata, 28th IEEE Symposium on the Foundations of Computer Science, 1987, pp. 78-88.

[V] L. G. Valiant, A Theory of the Learnable, Comm. ACM, 27(1984), pp. 1134-1142.

[V1] L. G. Valiant, Learning Disjunctions of

Conjunctions, 9th IJCAI, 1985, Vol. 1, pp. 560-566, Los Angeles, CA.

[V2] L. G. Valiant, Deductive Learning, Phil. Trans. R. Soc. Lond. A 312(1984), pp. 441-446.

[ZL] A.K. Zvonkin and L.A. Levin, The complexity of finite objects and development of the concepts of information and randomness by means of the theory of algorithms, *Russ. Math. Serv.* 25(1970), pp. 83-124.

7. Appendices

Appendix 1

Proof of Theorem A. First we consider the standard enumeration of all partial recursive functions $\phi_1, \phi_2, \ldots$. We change each ϕ into a partial recursive function ψ with the same range as ϕ but with, for each x, the value of $\psi(<x, k>)$ is defined only if $\psi(<x, 1>), \psi(<x, 2>), \ldots, \psi(<x, k-1>)$ are defined. (Assign values to arguments in enumeration order.) We use each ψ to define a enumerable function s by approximations $s^k(x)$, $k = 1, 2, \ldots$, from below:

$$s(x) = \sup\{s^k(x): s^k(x) = p/q, \psi(<x, k>) = <p, q>, k = 1,2,\ldots\}.$$

The resulting s-enumeration contains all enumerable functions. Next we use each enumerable function s to compute a semimeasure μ from below. Initially, set $\mu(x) = 0$ for all x. If $s(1)$ is undefined then μ will not change any more and it is trivially a semimeasure. Otherwise, for $k = 1, 2, \ldots$, if $s^k(1) + s^k(2) + \ldots + s^k(k) \leq 1$ then set $\mu(i) := s^k(i)$ for $i = 1, 2, \ldots, k$, else the computation of μ is finished.

There are three mutually exclusive ways the computation of μ can go, exhausting all possibilities. Firstly, s is already a semimeasure and $\mu := s$. Secondly, for some x and k with $x \leq k$ the value $s^k(x)$ is undefined. Then the values of μ do not change any more from $\mu(i) = s^{k-1}(i)$ for $i = 1, 2, \ldots, k-1$, and $\mu(i) = 0$ for $i \geq k$, even though the computation of μ goes on forever. Thirdly, there is a first k such that $s^k(1) + s^k(2) + \ldots + s^k(k) > 1$, that is, the new approximation of μ violates the condition of measure. Then the approximation of μ is finished as in the second case. But in this case the algorithm terminates, and μ is even recursive.

Thus, the above procedure yields an effective enumeration $\mu_1, \mu_2, \ldots$ of all enumerable semimeasures. Define the function μ_0 as:

$$\mu_0(x) = \sum_n 2^{-n} \mu_n(x).$$

It follows that μ_0 is a semimeasure since

$$\sum_x \mu_0(x) = \sum_n 2^{-n} \sum_x \mu_n(x) \leq \sum_n 2^{-n} = 1.$$

The function μ_0 is also enumerable, since $\mu_n(x)$ is enumerable in n and x. (Use the universal partial recursive function ϕ_0 and the construction above.) Finally, μ_0 dominates each μ_n since $\mu_0(x) > 2^{-n} \mu_n(x)$. Therefore, μ_0 is a universal enumerable semimeasure. Obviously, there are countably infinite universal enumerable semimeasures. We now fix a *reference* universal enumerable semimeasure and denote it by $\mathbf{m}(x)$. □

Appendix 2.

Proof of Theorem B. Since $2^{-K(x)}$ represents the contribution to $P_U(x)$ by a shortest program for x, $2^{-K(x)} \leq P_U(x)$ for all x. Since $P_U(x)$ is enumerable by enumerating all programs for x, we have by the universality of $\mathbf{m}(x)$ that there is a fixed constant c such that for all x we have $P_U(x) \leq c\,\mathbf{m}(x)$.

It remains to show that $\mathbf{m}(x) = O(2^{-K(x)})$. This is equivalent to showing that for some constant c we have $-\log \mathbf{m}(x) \geq K(x) + c$. It suffices to exhibit a prefix code such that for some other fixed constant c', for each x there is a code word p such that $l(p) \leq -\log \mathbf{m}(x) + c'$, together with a prefix-machine T such that $T(p) = x$. Then, $K_T(x) \leq l(p)$ and hence by the Invariance Theorem 1 also $K(x) \leq l(p)$ up to a fixed additive constant. First we recall a construction for the Shannon-Fano code.

Claim. If μ is a measure on the integers, $\sum_x \mu(x) \leq 1$, then there is a binary prefix-code $r: N \to \{0, 1\}^*$ such that the code words $r(1), r(2), \ldots$ are in lexicographical order, such that $l(r(x)) \leq -\log \mu(x) + 2$. This is the Shannon-Fano code.

Proof. Let $[0, 1)$ be the half open unit real interval. The half open interval $[0.x, 0.x + 2^{-l(x)})$ corresponding to the set (cylinder) of reals $\Gamma_x = \{0.y: y = x\,z\}$ (x finite and y and z infinite binary strings) is called a *binary interval.* We cut off disjoint, consecutive, adjacent (not necessarily binary) intervals I_n of length $\mu(n)$ from the left end of $[0, 1)$, $n = 1, 2, \ldots$. Let i_n be the length of the longest binary interval contained in I_n. Set $r(n)$ equal to the binary word corresponding to the first such interval. It is easy to see that I_n is covered by at most four binary intervals of length i_n, from which the claim follows. □

Since $\mathbf{m}(x)$ is enumerable, there is a partial recursive function $\phi(t, x)$ such that $\phi(t, x) \leq \mathbf{m}(x)$ for all t, and $\lim_{t \to \infty} \phi(t, x) = \mathbf{m}(x)$. Let $\psi(t, x) = 2^{-k}$, with k is a positive integer, be the greatest partial recursive lower bound of this form on $\phi(t, x)$. We can assume that ψ enumerates its range without repetition. Then,

$$\sum_{x,t} \psi(t, x) = \sum_x \sum_t \psi(t, x) \leq \sum_x 2\,\mathbf{m}(x) \leq 2.$$

(The series $\sum_t \psi(t, x)$ can only converge to precisely $2\,\mathbf{m}(x)$ in case there is a positive integer k such that $\mathbf{m}(x) = 2^{-k}$.)

Similar to before, we chop off consecutive, adjacent, disjoint half open intervals $I_{t,x}$ of length $\psi(t, x)/2$, in order of computation of $\psi(t, x)$, starting from the left side of $[0, 1)$. This is possible by the last displayed equation. It is easy to see that we can construct a prefix-machine T as follows. If Γ_p is the largest binary interval of $I_{t,x}$, then $T(p) = x$. Otherwise, $T(p)$ is undefined (e.g., T doesn't halt).

By construction of ψ, for each x there is a $\psi(t, x) > \mathbf{m}(x)/2$. By the construction in the Claim, each interval $I_{t,x}$ has length $\psi(t, x)/2$. Each I-interval contains a binary interval Γ_p of length at least one quarter of that of I. Therefore, there is a p with $T(p) = x$ such that $2^{-l(p)} \geqslant \mathbf{m}(x)/16$. This implies $K_T(x) \leqslant -\log \mathbf{m}(x) + 4$. The proof of the theorem is finished. □

The ITLI Prepublication Series

1986

86-01 The Institute of Language, Logic and Information
86-02 Peter van Emde Boas — A Semantical Model for Integration and Modularization of Rules
86-03 Johan van Benthem — Categorial Grammar and Lambda Calculus
86-04 Reinhard Muskens — A Relational Formulation of the Theory of Types
86-05 Kenneth A. Bowen, Dick de Jongh — Some Complete Logics for Branched Time, Part I Well-founded Time, Forward looking Operators
86-06 Johan van Benthem — Logical Syntax

1987

87-01 Jeroen Groenendijk, Martin Stokhof — Type shifting Rules and the Semantics of Interrogatives
87-02 Renate Bartsch — Frame Representations and Discourse Representations
87-03 Jan Willem Klop, Roel de Vrijer — Unique Normal Forms for Lambda Calculus with Surjective Pairing
87-04 Johan van Benthem — Polyadic quantifiers
87-05 Víctor Sánchez Valencia — Traditional Logicians and de Morgan's Example
87-06 Eleonore Oversteegen — Temporal Adverbials in the Two Track Theory of Time
87-07 Johan van Benthem — Categorial Grammar and Type Theory
87-08 Renate Bartsch — The Construction of Properties under Perspectives
87-09 Herman Hendriks — Type Change in Semantics: The Scope of Quantification and Coordination

1988

Logic, Semantics and Philosophy of Language:

LP-88-01 Michiel van Lambalgen — Algorithmic Information Theory
LP-88-02 Yde Venema — Expressiveness and Completeness of an Interval Tense Logic
LP-88-03 — Year Report 1987
LP-88-04 Reinhard Muskens — Going partial in Montague Grammar
LP-88-05 Johan van Benthem — Logical Constants across Varying Types
LP-88-06 Johan van Benthem — Semantic Parallels in Natural Language and Computation
LP-88-07 Renate Bartsch — Tenses, Aspects, and their Scopes in Discourse
LP-88-08 Jeroen Groenendijk, Martin Stokhof — Context and Information in Dynamic Semantics
LP-88-09 Theo M.V. Janssen — A mathematical model for the CAT framework of Eurotra
LP-88-10 Anneke Kleppe — A Blissymbolics Translation Program

Mathematical Logic and Foundations:

ML-88-01 Jaap van Oosten — Lifschitz' Realizabiility
ML-88-02 M.D.G. Swaen — The Arithmetical Fragment of Martin Löf's Type Theories with weak Σ-elimination
ML-88-03 Dick de Jongh, Frank Veltman — Provability Logics for Relative Interpretability
ML-88-04 A.S. Troelstra — On the Early History of Intuitionistic Logic
ML-88-05 A.S. Troelstra — Remarks on Intuitionism and the Philosophy of Mathematics

Computation and Complexity Theory:

CT-88-01 Ming Li, Paul M.B.Vitanyi — Two Decades of Applied Kolmogorov Complexity
CT-88-02 Michiel H.M. Smid — General Lower Bounds for the Partitioning of Range Trees
CT-88-03 Michiel H.M. Smid, Mark H. Overmars Leen Torenvliet, Peter van Emde Boas — Maintaining Multiple Representations of Dynamic Data Structures
CT-88-04 Dick de Jongh, Lex Hendriks Gerard R. Renardel de Lavalette — Computations in Fragments of Intuitionistic Propositional Logic
CT-88-05 Peter van Emde Boas — Machine Models and Simulations (revised version)
CT-88-06 Michiel H.M. Smid — A Data Structure for the Union-find Problem having good Single-Operation Complexity
CT-88-07 Johan van Benthem — Time, Logic and Computation
CT-88-08 Michiel H.M. Smid, Mark H. Overmars Leen Torenvliet, Peter van Emde Boas — Multiple Representations of Dynamic Data Structures
CT-88-09 Theo M.V. Janssen — Towards a Universal Parsing Algorithm for Functional Grammar
CT-88-10 Edith Spaan, Leen Torenvliet, Peter van Emde Boas — Nondeterminism, Fairness and a Fundamental Analogy
CT-88-11 Sieger van Denneheuvel, Peter van Emde Boas — Towards implementing RL

Other prepublications:

X-88-01 Marc Jumelet — On Solovay's Completeness Theorem

1989

Logic, Semantics and Philosophy of Language:

LP-89-01 Johan van Benthem — The Fine-Structure of Categorial Semantics
LP-89-02 Jeroen Groenendijk, Martin Stokhof — Dynamic Predicate Logic, towards a compositional, non-representational semantics of discourse
LP-89-03 Yde Venema — Two-dimensional Modal Logics for Relation Algebras and Temporal Logic of Intervals
LP-89-04 Johan van Benthem — Language in Action
LP-89-05 Johan van Benthem — Modal Logic as a Theory of Information
LP-89-06 Andreja Prijatelj — Intensional Lambek Calculi: Theory and Application

Mathematical Logic and Foundations:

ML-89-01 Dick de Jongh, Albert Visser — Explicit Fixed Points for Interpretability Logic
ML-89-02 Roel de Vrijer — Extending the Lambda Calculus with Surjective Pairing is conservative
ML-89-03 Dick de Jongh, Franco Montagna — Rosser Orderings and Free Variables
ML-89-04 Dick de Jongh, Marc Jumelet, Franco Montagna — On the Proof of Solovay's Theorem
ML-89-05 Rineke Verbrugge — Σ-completeness and Bounded Arithmetic
ML-89-06 Michiel van Lambalgen — The Axiomatization of Randomness

Computation and Complexity Theory:

CT-89-01 Michiel H.M. Smid — Dynamic Deferred Data Structures
CT-89-02 Peter van Emde Boas — Machine Models and Simulations
CT-89-03 Ming Li, Herman Neuféglise, Leen Torenvliet, Peter van Emde Boas — On Space efficient Simulations
CT-89-04 Harry Buhrman, Leen Torenvliet — A Comparison of Reductions on Nondeterministic Space
CT-89-05 Pieter H. Hartel, Michiel H.M. Smid Leen Torenvliet, Willem G. Vree — A Parallel Functional Implementation of Range Queries
CT-89-06 H.W. Lenstra, Jr. — Finding Isomorphisms between Finite Fields
CT-89-07 Ming Li, Paul M.B. Vitanyi — A Theory of Learning Simple Concepts under Simple Distributions and Average Case Complexity for the Universal Distribution (Prel. Version)

Other prepublications:

X-89-01 Marianne Kalsbeek — An Orey Sentence for Predicative Arithmetic
X-89-02 G. Wagemakers — New Foundations: a Survey of Quine's Set Theory
X-89-03 A.S. Troelstra — Index of the Heyting Nachlass
X-89-04 Jeroen Groenendijk, Martin Stokhof — Dynamic Montague Grammar, a first sketch
X-89-05 Maarten de Rijke — The Modal Theory of Inequality