Philosophical Applications of Computational Learning Theory:

Chomskyan Innateness and Occam's Razor

Afstudeerscriptie Filosofie van de Informatica (15sp)

R. M. de Wolf
114903

Erasmus Universiteit Rotterdam
Vakgroep: Theoretische Filosofie
Begeleider: Dr. G. J. C. Lokhorst

Juli 1997

Contents

Preface

What is learning? Learning is what makes us adapt to changes and threats, and what allows us to cope with a world in flux. In short, learning is what keeps us alive. Learning has strong links to almost any other topic in philosophy: scientific inference, knowledge, truth, reasoning (logic), language, anthropology, behaviour (ethics), good taste (aesthetics), and so on. Accordingly, it can be seen as one of the quintessential philosophical topics—an appropriate topic for a graduation thesis! Much can be said about learning, too much to fit in a single thesis. Therefore this thesis is restricted in scope, dealing only with *computational learning theory* (often abbreviated to COLT).

Learning seems so simple: we do it every day, often without noticing it. Nevertheless, it is obvious that some fairly complex *mechanisms* must be at work when we learn. COLT is the branch of Artificial Intelligence that deals with the computational properties and limitations of such mechanisms. The field can be seen as the intersection of Machine Learning and complexity theory. COLT is a very young field—the publication of Valiant's seminal paper in 1984 may be seen as its birth—and it appears that virtually none of its many interesting results are known to philosophers. Some philosophical work has referred to complexity theory (for instance [Che86]) and some has referred to Machine Learning (for instance [Tha90]), but as far as I know, thus far no philosophical use has been made of the results that have sprung from COLT. For instance, at the time of writing of this thesis, the Philosopher's Index, a database containing most major publications in philosophical journals or books, contains no entries whatsoever that refer to COLT or to its main model, the model of PAC learning; there is hardly any reference to Kolmogorov complexity, either. Stuart Russell devotes two pages to PAC learning [Rus91, pp. 43–44] and James McAllister devotes one page to Kolmogorov complexity as a quantitative measure of simplicity [McA96, pp. 119–120], but both do not provide more than a sketchy and superficial explanation.

As its title already indicates, the present thesis tries to make contact between COLT and philosophy. The aim of the thesis is threefold. The first and most shallow goal is to obtain a degree in philosophy for its author. The second goal is to take a number of recent results from computational learning theory, insert them in their appropriate philosophical context, and see how they bear on various ongoing philosophical discussions. The third and most ambitious goal is to draw the attention of philosophers to computational learning theory in general. Unfortunately, the traditional reluctance of philosophers (in particular those of a non-analytical strand) to use formal methods, as well as their inapt-

ness with formal methods once they have outgrown that reluctance, makes this goal rather hard to attain. Nevertheless, it is my opinion that computational learning theory—or, for that matter, complexity theory as a whole—has much to offer to philosophy. Accordingly, this thesis may be seen as a plea for the philosophical relevance of computational learning theory as a whole.

The thesis is organized as follows. In the first chapter, we will give an overview of the main learning setting used in COLT. We will stick to the rigorous formal definitions as used in COLT, but supplement them with a lot of informal and intuitive comment in order to make them accessible and readable for philosophers. After that introductory chapter, the second chapter applies results from COLT to Noam Chomsky's ideas about *language learning* and the innateness of linguistic biases. The third chapter gives an introduction to the theory of *Kolmogorov complexity*, which provides us with a fundamental measure of simplicity. Kolmogorov complexity does not belong to computational learning theory proper (its invention in fact pre-dates COLT), but its main application for us lies in *Occam's Razor*. This fundamental maxim tells us to go for the simplest theories consistent with the data, and is highly relevant in the context of learning in general, and scientific theory construction in particular. The fourth chapter provides different formal settings in which some form of the razor can be mathematically justified, using Kolmogorov complexity to quantitatively measure simplicity. Finally, we end with a brief chapter that summarizes the thesis in non-technical terms.

Let me end this preface by expressing my gratitude to a number of people who contributed considerably to the contents of this thesis. First of all, I would of course like to thank my thesis advisor Gert-Jan Lokhorst—one of those philosophers who, like myself, do not hesitate to invoke formal definitions and results whenever these might be useful—for his many comments and suggestions. Secondly, many thanks should go to Shan-Hwei Nienhuys-Cheng, who put me on the track of inductive learning in the first place, by inviting me to join her in writing a book on inductive logic programming [NW97]—a book which eventually took us more than two and a half years to finish. Chapter 18 of that book actually provided the basis for a large part of the first chapter of the present thesis. Finally, I would like to thank Jeroen van Rijen for his many helpful comments, Peter Sas and Peter Grünwald for some references on linguistics, and Paul Vitányi for his course on learning and Kolmogorov complexity.

Chapter 1

Introduction to Computational Learning Theory

1.1 Introduction

This thesis is about philosophical applications of "computational learning theory", and the present chapter provides an introduction to this field.

Why should we, as philosophers, be interested in something like a theory of learning? The importance of learning can be illustrated on the basis of the following quotation from Homer's Iliad:

> Him she found sweating with toil as he moved to and fro about his bellows in eager haste; for he was fashioning tripods, twenty in all, to stand around the wall of his well-builded hall, and golden wheels had he set beneath the base of each that of themselves they might enter the gathering of the gods at his wish and again return to his house, a wonder to behold.
> *Iliad*, XVIII, 372–377 (pp. 315–317 of [Hom24], second volume).

This quotation might well be the first ever reference to something like *Artificial Intelligence*: man-made (or in this case, god-made) artifacts displaying intelligent behaviour. As Thetis, Achilles' mother, enters Hephaestus' house in order to fetch her son a new armour, she finds Hephaestus constructing something we today would call *robots*. His twenty tripods are *of themselves* to serve the gathering of the gods (bring them food, etc.), whenever Hephaestus so desires.

Let us consider for a moment the kind of behaviour such a tripod should display. Obviously, it should be able to recognise Hephaestus' voice, and to extract his wishes from his words. But furthermore, when serving the gods, the tripod should "know" and act upon many requirements, such as the following:

1. If there is roasted owl for dinner, don't give any to Pallas Athena.
2. Don't come too close to Hera if Zeus has committed adultery again.
3. Stop fetching wine for Dionysus when he is too drunk.

   ...

It is clear that this list can be continued without end. Again and again, one can think of new situations that the tripod should be able to adapt to properly. It seems impossible to take all these requirements into account explicitly in the construction of the intelligent tripod. The task of "coding" each of the infinite number of requirements into the tripods may be considered too much, even for Hephaestus, certainly one of the most industrious among the Greek gods.

One solution to this problem would be to initially endow the tripod with a modest amount of general knowledge about what it should do, and to give it the ability to *learn* from the way the environment reacts to its behaviour. That is, if the tripod does something wrong, it can adjust its knowledge and its behaviour accordingly, thus avoiding to make the same mistake in the future.[1] In that way, the tripod need not know everything beforehand. Instead, it can build up most of the required knowledge along the way. Thus the tripod's ability to learn would save Hephaestus a lot of trouble.

The importance of learning is not restricted to artifacts built to serve divine wishes. Human beings, from birth till death, engage in an ongoing learning process. After all, children are not born with their native language, polite manners, the abilities to read and write, earn their living, make jokes, or to do philosophy or science (or even philosophy *of* science). These things have to be *learned*. In fact, if we take 'learning' in a sufficiently broad sense, any kind of adaptive behaviour will fall under the term. Since learning is of crucial importance to us, a *theory* of learning is of crucial importance to philosophy.

1.2 Algorithms that Learn Concepts from Examples

After the previous section, the importance of a theory of learning should be clear. But why *computational* learning theory? Well, a theory of learning should be about the way we human beings learn, or, more generally, about the way any kind of learning can be achieved. A description of "a way of learning" will usually boil down to something like a "recipe": learning amounts to doing such-and-such things, taking such-and-such steps. Now, it seems that the only *precise* way we have to specify this "such and such steps"—and of course we should aim for precision—is by means of an *algorithm*. In general, an algorithm, or a mechanical procedure, precisely prescribes the sequence of steps that have to be taken in order to solve some problem. Hence it is obvious that learning theory can (or perhaps even *should*) involve the study of *algorithms* that learn.[2] Since algorithms perform computation, algorithmic theory of learning is usually called *computational learning theory*.

In this thesis, we will mainly be concerned with learning a *concept* from

[1] Of course, for this scheme to work, we have to assume that the tripod "survives" its initial failures. If Zeus immediately smashes the tripod into pieces for bringing him white instead of red wine, the tripod won't be able to learn from its experience.

[2] The notion of an algorithm includes *neural networks*, at least those that can be simulated on an ordinary computer. In fact, the learnability of neural networks has been one of the most prominent research areas in computational learning theory.

examples. If we take the term 'example' in a sufficiently broad sense, almost any kind of learning will be based on examples, so restricting attention to learning *from examples* is not really a restriction. On the other hand, restricting attention to learning a *concept* appears to be quite restrictive, since it excludes learning "know-how" knowledge, for instance learning how to run a marathon. However, many cases of "know-how" learning can actually quite well be modeled or redescribed as cases of concept learning. For instance, learning a language may at first sight appear to be a case of learning know-how (i.e., knowing how to use words), but it can also be modeled as the learning of a *grammar* for a language.[3] Accordingly, concept learning can be used to model any kind of learning in which the learned "thing" can feasibly be represented by some mathematical construct—a grammar, a logical formula, a neural network, and what not. This includes a very wide range of topics, from language learning to large parts of empirical science.

Induction, which is the usual name for learning from examples, has been a topic of inquiry for centuries. The study of induction can be approached from many angles. Like most other scientific disciplines, it started out as a part of philosophy. Philosophers particularly focused on the role induction plays in the empirical sciences. For instance, Aristotle characterized science roughly as deduction from first principles, which were to be obtained by means of induction from experience [Ari60]. (Though it should be noted that Aristotle's notion of induction was rather different from the modern one, involving the "seeing" of the "essential forms" of examples.)

After the Middle Ages, Francis Bacon [Bac94] revived the importance of induction from experience (in the modern sense) as the main scientific activity. In later centuries, induction was taken up by many philosophers. David Hume [Hum56, Hum61] formulated what is nowadays called the 'problem of induction', or 'Hume's problem': how can induction from a finite number of cases result in knowledge about the infinity of cases to which an induced general rule applies? What justifies inferring a general rule, or "law of nature", from a finite number of cases? Surprisingly, Hume's answer was that there is *no* such justification. In his view, it is simply a psychological fact about humans beings that when we observe some particular pattern recur in different cases (without observing counterexamples to the pattern), we tend to expect this pattern to appear in all similar cases. In Hume's view, this inductive expectation is a *habit*, analogous to the habit of a dog who runs to the door after hearing his master call, expecting to be let out. Later philosphers such as John Stuart Mill [Mil58] tried to answer Hume's problem by stating conditions under which an inductive inference is justified. Other philosophers who made important comments on induction were Stanley Jevons [Jev74] and Charles Sanders Peirce [Pei58].

In our century, induction was mainly discussed by philosophers and mathematicians who were also involved in the development and application of formal logic. Their treatment of induction was often in terms of the probability or the "degree of confirmation" that a particular theory or hypothesis receives from available empirical data. Some of the main contributors are Bertrand

[3] This is what the Chomskyan revolution in linguistics is all about, see the next chapter.

Russell [Rus80, Rus48], Rudolf Carnap [Car52, Car50], Carl Hempel [Hem45a, Hem45b, Hem66], Hans Reichenbach [Rei49], and Nelson Goodman [Goo83]. Particularly in Goodman's work, an increasing number of unexpected conceptual problems appeared for induction. In the 1950s and 1960s, induction was sworn off by philosophers of science such as Karl Popper [Pop59].[4]

However, in roughly those same years, it was recognised in the rapidly expanding field of Artificial Intelligence that the knowledge an AI system needs to perform its tasks, should not all be hand-coded into the system beforehand. Instead, it is much more efficient to provide the system with a relatively small amount of knowledge and with the ability to adapt itself to the situations it encounters—to *learn* from its experience. Thus the study of induction switched from philosophy to Artificial Intelligence. The branch of AI that studies learning is called *Machine Learning*. As Marvin Minsky, one of the founders of AI, wrote: "Artificial Intelligence is the science of making machines do things that would require intelligence if done by man" [Min68, p. v]. Given this view, the study of induction is indeed part of AI, since learning from examples certainly requires intelligence if done by man.

1.3 The PAC Model and Efficiency

Since Machine Learning is concerned with formal learning *algorithms*, it needs formal models of what it means to learn something: what kinds of "examples" and other resources does a learning algorithm have at its disposal, and what are its goals? In general, a learning algorithm reads a number of examples for some unknown *target* concept, and has to induce or learn some concept on the basis of these examples. Initial analysis of learnability in Machine Learning was mainly done in terms of Gold's paradigm of *identification in the limit* [Gol67], but nowadays Valiant's paradigm of *PAC learnability* [Val84] is usually considered to provide a better model of learnability. A PAC algorithm is an algorithm that reads examples concerning some target concept, which is taken from some class $\mathcal{F}$ of concepts. The algorithm knows from which class the target concept is chosen, but it does not know which *particular* concept is the target, and its only access to the target is through the examples it reads. These examples will usually not provide complete knowledge of the target concept, so we cannot expect our algorithm to learn the target exactly. Instead, we can only hope to learn an *approximately correct* concept: a concept which diverges only slightly from the target. Moreover, since the given set of examples may be biased and need not always be a good representative of the target concept as a whole, we cannot even expect to learn approximately correctly every time. Accordingly, the best our algorithm can do, is learn a concept which is *probably* approximately correct (PAC) with respect to the target concept, whenever the target is drawn from $\mathcal{F}$. That is, a PAC algorithm should, with high probability, learn a concept which diverges only slightly from the target concept.

[4] Interestingly enough, Thomas Kuhn, Popper's antipode in the philosophy of science, later became involved in computer models of inductive concept learning from examples. See pp. 474–482 of [Kuh77].

In the PAC model, a class of concepts is only considered learnable if there is an *efficient* PAC algorithm for that class.[5] Efficiency concerns two major complexity issues: how many *examples* do we need to ensure that we will probably find an approximately correct concept (*sample complexity*), and how many *steps* do we need to take to find such a concept (*time complexity*)? An algorithm is efficient if both its sample and its time complexity can be upper-bounded by a *polynomial* function in the inputs of the algorithm.[6]

Before embarking on formal expositions of PAC algorithms and their sample and time complexity, let us first say something about why we call an algorithm with polynomially-bounded running time an *efficient* algorithm. Suppose we want to solve some family of problems, and we can measure the size of each particular problem (or *instance*) in that family by some integer n. For instance, we might want to construct an algorithm for solving the *traveling salesman problem* (TSP): given a road map and a number of cities on the map, find the shortest route that leads past each city on the map. For simplicity, let us define the size of a particular traveling salesman problem as the number n of cities on the map. Suppose we have two algorithms for solving TSP: each takes a particular TSP-instance as input, and finds the shortest route for us in a finite number of steps. Now, suppose the first algorithm, when given a problem of size n as input, gives the right answer after n^2 steps, while the second needs 2^n steps.[7] Let us call the first the *polynomial* algorithm, and the second the *exponential* algorithm. Consider the number of steps needed by these two algorithms for larger n:

Number of cities:	1	5	10	50	100	...
No. of steps (polynomial):	1	25	100	2500	10000	...
No. of steps (exponential):	2	32	1024	$1.13 \cdot 10^{15}$	$1.27 \cdot 10^{30}$	...

As can be seen from this table, the time required by the exponential algorithm really explodes for larger n, while for the polynomial algorithm it grows much more moderately. Both algorithms solve the same problem correctly, but the polynomial algorithm needs much less time for this than the exponential one.

The relative efficiency of the polynomial algorithm can also be seen in another way. According to Moore's well known law, the speed of computers doubles every one and a half years. Suppose that in Januari 1996 we can solve TSP's of length up to n_1 in one hour's time using the polynomial algorithm, and TSP's of length up to n_2 with the exponential algorithm. Now suppose computing power doubles, and in July 1997 we have a computer which can make

[5] Note carefully that learnability is a property of *classes* of concepts, rather than of individual concepts. A class consisting of a single concept f is always learnable, because a learning algorithm can already know in advance that the target concept has to be f in this case.

[6] A *polynomial* function with variables $x_1, \ldots, x_n$ is a sum of terms of the form $cx_1^{e_1}x_2^{e_2}\ldots x_n^{e_n}$, where c is an arbitrary real constant, and the exponents $e_1, \ldots, e_n$ are non-negative real constants. For instance, $x^2 + 3$ and $3x_1x_2 + 5x_2^2$ are polynomials.

[7] In fact, it is a big open question whether TSP can be solved by a polynomial-time algorithm, because this problem, in a slightly different form, is known to be $\mathcal{NP}$-complete. See note 16 on p. 39 for more on $\mathcal{NP}$-completeness.

twice as many steps in one hour as the computer we used in Januari 1996. Then it is easy to show that with this faster computer, we can solve TSP's of length up to $\sqrt{2} \cdot n_1 \approx 1.41 \cdot n_1$ with the polynomial algorithm: an improvement of about 41%, which is quite nice. However, with the exponential algorithm, we can only solve problems of length up to $n_2 + 1$ in one hour! Thus an increase in computing power makes a great difference if we use the efficient polynomial algorithm, but makes hardly any difference using the exponential algorithm.

1.4 PAC Algorithms

Let us first illustrate and motivate the definition of a PAC algorithm by means of a metaphorical example. Suppose some biology student wants to learn from examples to distinguish insects from other animals. That is, he wants to learn the concept of an 'insect' within the domain of all animals. A teacher gives the student examples: a positive example is an insect, a negative example is some other animal. The student has to develop his own concept of what an insect is on the basis of these examples. Now, the student will be said to have learned the concept *approximately correctly*, if, when afterwards tested, he classifies only a small percentage of given test animals incorrectly as insect or non-insect. In other words, his own developed concept should not diverge too far from the real concept of an 'insect'.

In the interest of fairness, we require that the animals given as examples during the learning phase, and the animals given afterwards as test, are all selected *by the same teacher* (or at least by teachers with the same inclinations). For suppose the student learns from a teacher with a particular interest in European animals, whose examples are mainly European animals. Then it would be somewhat unfair if the animals that were given afterwards to test the student, were selected by a different teacher having a decisive interest in the very different set of African insects. In other words: the student should be taught and tested by the same teacher.

Let us now formalize this setting. To my knowledge, three different textbooks for computational learning theory exist to date, respectively written by Natarajan [Nat91], Anthony and Biggs [AB92], and Kearns and Vazirani [KV94]. The formal definitions in this chapter will mostly follow Natarajan.

Definition 1.1 A *domain* X is a set of strings over some finite alphabet Σ. The *length* of some $x \in X$ is the string length of x. $X^{[n]}$ denotes the set of all strings in X of length at most n.

A *concept* f is a subset of X, a *concept class* $\mathcal{F}$ is a set of concepts. An *example* for f is a pair (x, y), where $x \in X$, y is called the *label* of the example, $y = 1$ if $x \in f$ and $y = 0$ otherwise. If $y = 1$ then the example is *positive*, if $y = 0$ it is *negative*.

If f and g are two concepts, then $f \Delta g$ denotes the *symmetric difference* of f and g: $f \Delta g = (f \backslash g) \cup (g \backslash f)$. $\diamond$

In our metaphor, X would be the set of descriptions of all animals, the target concept $f \subseteq X$ would be the set of descriptions of all insects, and the student

would develop his own concept $g \subseteq X$ on the basis of a number of positive and negative examples (i.e., insects and non-insects). The symmetric difference $f \Delta g$ would be the set of all animals which the student classifies incorrectly: all insects that he takes to be non-insects and all non-insects he takes to be insects.

For technical reasons, we restrict the examples to those of length at most some number n, so all examples are drawn from $X^{[n]}$. Note that $X^{[n]}$ is a finite set. We assume these examples are given according to some unknown probability distribution $\mathbf{P}$ on $X^{[n]}$, which reflects the particular interests of the teacher. If $S \subseteq X^{[n]}$, we let $\mathbf{P}(S)$ denote the probability that a member of $X^{[n]}$ that is drawn according to $\mathbf{P}$, is a member of S (i.e., $\mathbf{P}(S) = \sum_{s \in S} \mathbf{P}(s)$). Now suppose the student has developed a certain concept g. Then in the test phase, he will misclassify some object $x \in X^{[n]}$ iff[8] $x \in f \Delta g$. Thus we can say that g is *approximately correct* if the probability that such a misclassified object is given during the test phase, is small:

$$\mathbf{P}(f \Delta g) \leq \varepsilon,$$

where $\varepsilon \in (0, 1]$ is called the *error* parameter. For instance, if $\varepsilon = 0.05$, then there is a chance of at most 5% that an arbitrary given test object from $X^{[n]}$ will be classified incorrectly. Note that the set of examples that is given, as well as the evaluation of approximate correctness of the learned concept g, depends on the *same* probability distribution $\mathbf{P}$. This formally reflects the fairness-requirement that the student is taught and tested by the same teacher.

After all these preliminaries, we can now define a *PAC algorithm* as an algorithm which, under some unknown distribution $\mathbf{P}$ and target concept f, learns a concept g which is *probably* approximately correct. 'Probably' here means with probability at least $1 - \delta$, where $\delta \in (0, 1]$ is called the *confidence parameter*. For instance, if $\delta = 0.1$ and the algorithm would be run an infinite number of times, at least 90% of these runs would output an approximately correct concept. The constants ε, δ, and n are given by the user as input to the algorithm.

Definition 1.2 A learning algorithm L is a *PAC algorithm* for a concept class $\mathcal{F}$ over domain X if

1. L takes as input real numbers $\varepsilon, \delta > 0$ and a natural number $n \in \mathbf{N}$, where ε is the *error* parameter, δ is the *confidence* parameter, and n is the *length* parameter.
2. L may call the procedure EXAMPLE, each call of which returns an example for some unknown *target* concept $f \in \mathcal{F}$ according to an arbitrary and unknown probability distribution $\mathbf{P}$ on $X^{[n]}$.
3. For all concepts $f \in \mathcal{F}$ and all probability distributions $\mathbf{P}$ on $X^{[n]}$, L outputs a concept g, such that with probability at least $1-\delta$, $\mathbf{P}(f \Delta g) \leq \varepsilon$.

◇

A PAC algorithm may be *randomized*, which means, informally, that it may "toss coins" and use the results in its computations. One further technicality:

[8] 'Iff' abbreviates 'if, and only if'.

a PAC algorithm should be *admissible*, meaning that for any input ε, δ, n, for any sequence of examples that EXAMPLE may return, and for any concept g, the probability that L outputs g should be well defined.

1.5 Sample Complexity

Having a PAC algorithm for a concept class $\mathcal{F}$ is nice, but having an *efficient* PAC algorithm for $\mathcal{F}$ is even nicer. In this section we analyze this efficiency in terms of the *number of examples* the algorithm needs (the *sample complexity*), while in the next section we treat the *number of steps* the algorithm needs to take (*time complexity*).

The sample complexity of a learning algorithm can be seen as a function from its inputs ε, δ, and n, to the maximum number of examples that the algorithm reads when learning an unknown target concept under an unknown probability distribution. Since the examples are drawn according to a probability distribution, different runs of the same algorithm with the same input and the same target concept and distribution may still read different examples. Thus different runs of the same algorithm with the same input may need a different number of examples in order to find a satisfactory concept. Therefore, the sample complexity as defined below relates to the maximum number of examples *over all runs* of the algorithm with the same input.

Definition 1.3 Let L be a learning algorithm for concept class $\mathcal{F}$. The *sample complexity* of L is a function s, with parameters ε, δ and n. It returns the maximum number of calls of EXAMPLE made by L, for all runs of L with inputs ε, δ, n, for all $f \in \mathcal{F}$ and all $\mathbf{P}$ on $X^{[n]}$. If no finite maximum exists, we let $s(\varepsilon, \delta, n) = \infty$. ◇

Of course, for the sake of efficiency we want this complexity to be as small as possible. A concept class is usually considered to be efficiently PAC learnable—as far as the required number of examples is concerned—if there is a PAC algorithm for this class for which the sample complexity is bounded from above by a polynomial function in $1/\varepsilon$, $1/\delta$, and n. Of course, even polynomials may grow rather fast (consider n^{100}), but still their growth rate is much more moderate than, for instance, exponential functions.

Definition 1.4 A concept class $\mathcal{F}$ is called *polynomial-sample PAC learnable*, if a PAC algorithm exists for f, which has a sample complexity bounded from above by a polynomial in $1/\varepsilon$, $1/\delta$, and n. ◇

Note that polynomial-sample PAC learnability has to do with the *worst case*: if the worst case cannot be bounded by a polynomial, a concept class is not polynomial-sample PAC learnable, even though there may be PAC algorithms which take only a small polynomial number of examples on average.

A crucial notion in the study of sample complexity is the dimension named after Vapnik and Chervonenkis [VC71]:

Definition 1.5 Let $\mathcal{F}$ be a concept class on domain X. We say that $\mathcal{F}$ *shatters* a set $S \subseteq X$, if $\{f \cap S \mid f \in \mathcal{F}\} = 2^S$, i.e., if for every subset S' of S, there is an $f \in \mathcal{F}$ such that $f \cap S = S'$. ◇

Definition 1.6 Let $\mathcal{F}$ be a concept class on domain X. The *Vapnik-Chervonenkis dimension* (VC-dimension) of $\mathcal{F}$, denoted by $\mathbf{D}_{VC}(\mathcal{F})$, is the greatest integer d such that there exists a set $S \subseteq X$ with $|S| = d$, that is shattered by $\mathcal{F}$. $\mathbf{D}_{VC}(\mathcal{F}) = \infty$ if no greatest d exists. $\diamond$

Note that if $\mathcal{F} = 2^S$, then $\mathcal{F}$ shatters S. Thus if $\mathcal{F} = 2^S$ for some finite set S, then $\mathcal{F}$ has $|S|$ as VC-dimension.

Example 1.1 Let $X = \{1,2,3,4\}$ and $\mathcal{F} = \{\{1\},\{2\},\{3\},\{4\},\{1,2\},\{2,3\},\{1,3,4\},\{1,2,3,4\}\}$ be a concept class. Then $\mathcal{F}$ shatters the set $S = \{1,2\}$, because $\{f \cap S \mid f \in \mathcal{F}\} = \{\emptyset,\{1\},\{2\},\{1,2\}\} = 2^S$. Thus $\mathcal{F}$'s "shattering" of S intuitively means that $\mathcal{F}$ "breaks" S into all possible pieces.

$\mathcal{F}$ also shatters $S' = \{1,2,3\}$, because $\{f \cap S' \mid f \in \mathcal{F}\} = \{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{2,3\},\{1,3\},\{1,2,3\}\} = 2^S$. $\mathcal{F}$ does not shatter $S'' = \{1,2,3,4\}$, since there is for instance no $f \in \mathcal{F}$ with $f \cap S'' = \{1,4\}$. In general, there is no set of four or more elements that is shattered by $\mathcal{F}$, so we have $\mathbf{D}_{VC}(\mathcal{F}) = |S'| = 3$. $\triangleleft$

Since we are actually dealing with $X^{[n]}$ rather than with X itself, we need the following definitions, which "project" the VC-dimension on $X^{[n]}$.

Definition 1.7 The *projection* of a concept f on $X^{[n]}$ is $f^{[n]} = f \cap X^{[n]}$. The *projection* of a concept class $\mathcal{F}$ on $X^{[n]}$ is $\mathcal{F}^{[n]} = \{f^{[n]} \mid f \in \mathcal{F}\}$. $\diamond$

Definition 1.8 Let $\mathcal{F}$ be a concept class on domain X. $\mathcal{F}$ is *of polynomial VC-dimension* if $\mathbf{D}_{VC}(\mathcal{F}^{[n]})$ is bounded from above by some polynomial in n. $\diamond$

The following fundamental result, due to Blumer, Ehrenfeucht, Haussler, and Warmuth [BEHW89], states the relation between polynomial-sample PAC learnability and the VC-dimension. For a proof we refer to Theorem 2.3 of [Nat91].

Theorem 1.1 *Let $\mathcal{F}$ be a concept class on domain X. Then $\mathcal{F}$ is polynomial-sample PAC learnable iff $\mathcal{F}$ is of polynomial VC-dimension.*

In the proof of the theorem, the following important lemma is proved (Lemma 2.1 of [Nat91]):

Lemma 1.1 *Let $\mathcal{F}$ be a concept class on a finite domain X. If $d = \mathbf{D}_{VC}(\mathcal{F})$, then*

$$2^d \leq |\mathcal{F}| \leq (|X|+1)^d.$$

Let us use this to obtain bounds on $|\mathcal{F}^{[n]}|$. Suppose the length parameter n is given, and the domain X is built from an alphabet Σ which contains $s \geq 2$ characters. Since we are only concerned with elements of the domain of length at most n, and the number of strings of length i over Σ is s^i, we can upper bound $|X|$ as follows:

$$|X| \leq s^0 + s^1 + \ldots + s^n = s^{n+1} - 1.$$

Putting $d = \mathbf{D}_{VC}(\mathcal{F}^{[n]})$ and substituting our upper bound on $|X|$ in the relation of the lemma, we obtain the following relation:

$$2^d \leq |\mathcal{F}^{[n]}| \leq s^{(n+1)d}.$$

If we take logarithms (with base 2) on both sides, using that $\log a^b = b \log a$ and $\log 2 = 1$, we obtain

$$d \leq \log |\mathcal{F}^{[n]}| \leq (n+1)d \log s.$$

This implies that a concept class $\mathcal{F}$ is of polynomial VC-dimension (i.e., d is bounded by a polynomial in n) iff $\log |\mathcal{F}^{[n]}|$ can be bounded by some polynomial $p(n)$ iff $|\mathcal{F}^{[n]}|$ can be bounded by $2^{p(n)}$. Thus if we are able to show that $|\mathcal{F}^{[n]}|$ is bounded in this way, we have thereby shown it to be polynomial-sample PAC learnable. And conversely, if $|\mathcal{F}^{[n]}|$ grows faster than $2^{p(n)}$, for any polynomial $p(n)$, then $\mathcal{F}$ is not polynomial-sample PAC learnable.

1.6 Time Complexity

In outline, the analysis of *time complexity* is similar to the analysis of sample complexity: the time complexity of a learning algorithm is a function from its inputs to the maximum number of computational steps the algorithm takes on these inputs. Here we assume that the procedure EXAMPLE takes at most some fixed constant number of steps. Again, we are mainly interested in the existence of algorithms which have a polynomially-bounded time complexity.

1.6.1 Representations

Unfortunately, things are somewhat more complicated than in the last section: the "number of examples" that an algorithm needs is unambiguous, but what about the "number of computational steps"? What counts as a computational step? In order to make this notion precise, we have to turn to some precise model of computation, where it is clear what a single step is. Usually Turing machines are used for this.[9] We will not go into details, but will just note here that a Turing machine programmed to learn some concept will often not be able to output the learned concept g itself efficiently, because this concept may be too large or even infinite. Therefore, instead of the concept g itself, the Turing machine will have to output some finite representation of g, which we call a *name* of g. Abstractly, a *representation* specifies the relation between concepts and their names:

Definition 1.9 Let $\mathcal{F}$ be a concept class, and Σ a set of symbols. Σ^+ denotes the set of all finite non-empty strings over Σ. A *representation* of $\mathcal{F}$ is a function $R : \mathcal{F} \to 2^{\Sigma^+}$, where we require that for each $f \in \mathcal{F}$, $R(f) \neq \emptyset$ and for every

[9] See [HU79, BJ89] for an introduction into Turing machines. Since Turing machines cannot represent arbitrary real numbers, we have to restrict the parameters δ and ε somewhat, for instance by only allowing them to be the inverses of integers.

distinct $f, g \in \mathcal{F}$, $R(f) \cap R(g) = \emptyset$. For each $f \in \mathcal{F}$, $R(f)$ is the set of *names* of f in R.

The *length* of a name $r \in R(f)$ is simply the string length of r, i.e., the number of symbols in r. The *size* of f in R is the length of the shortest name in $R(f)$, denoted by $l_{min}(f, R)$. ◇

The requirement that $R(f) \neq \emptyset$ for each $f \in \mathcal{F}$ means that each concept in $\mathcal{F}$ has at least one name, while $R(f) \cap R(g) = \emptyset$ for every distinct f, g means that no two distinct concepts share the same name. Note the difference between the string length of a string $x \in X$ and the size of a concept $f \in \mathcal{F}$ in R: the latter depends on R, the former does not.

The aim of the analysis of time complexity is to be able to bound by a polynomial function the number of steps needed for learning. After all, we are interested in *efficient* learning. However, if a learning algorithm provided us with a name of an approximately correct concept in a polynomial number of steps, but we were not able to decide in polynomial time whether that concept actually contains a given $x \in X$, we still had a computational problem. Therefore, a representation R should be *polynomially evaluable*: given an $x \in X$ and a name r of a concept f, we should be able to find out, in polynomial time, whether $x \in f$. This is defined as follows.

Definition 1.10 Let R be a representation of a concept class $\mathcal{F}$ over domain X. We say that R is *evaluable* if there exists an algorithm which, for any $f \in \mathcal{F}$, takes any $x \in X$ and any name $r \in R(f)$ as input, and decides in a finite number of steps whether $x \in f$. R is *polynomially evaluable* if there is such an algorithm, whose running time is bounded by a polynomial in the lengths of x and r. ◇

In the sequel, whenever we write 'representation' we actually mean a *polynomially evaluable* representation.

1.6.2 Polynomial-Time PAC Learnability

In order to be able to study time complexity, we need to change the definition of a PAC learning algorithm somewhat to incorporate the representation: a PAC algorithm for a concept class $\mathcal{F}$ *in representation* R should output the *name* of a concept g, rather than g itself.

Now time complexity can be defined as follows, where we introduce a new parameter l that bounds the size of the concepts considered:

Definition 1.11 Let L be a learning algorithm for concept class $\mathcal{F}$ in representation R. The *time complexity* of L is a function t, with parameters ε, δ, n, and l. It returns the maximum number of computational steps made by L, for all runs of L with inputs $\varepsilon, \delta, n, l$, for all $f \in \mathcal{F}$ such that $l_{min}(f, R) \leq l$, and all $\mathbf{P}$ on $X^{[n]}$. If no finite maximum exists, we define $t(\varepsilon, \delta, n, l) = \infty$. ◇

Definition 1.12 A concept class $\mathcal{F}$ is called *polynomial-time PAC learnable* in a representation R, if a PAC algorithm exists for f in R, which has a time complexity bounded from above by a polynomial in $1/\varepsilon$, $1/\delta$, n, and l. ◇

Let us suppose we have some concept class $\mathcal{F}$ of polynomial VC-dimension. Then we know $\mathcal{F}$ is polynomial-sample PAC learnable, so we only need a polynomial number of examples. Now, in order to achieve polynomial-time PAC learnability of $\mathcal{F}$, it is sufficient to have an algorithm that finds, in a polynomial number of steps, a concept that is *consistent* with the given examples. A concept is consistent if it contains all given positive examples and none of the given negative examples.

Definition 1.13 Let g be a concept and S be a set of examples. We say g is *consistent* with S, if $x \in g$ for every $(x, 1) \in S$ and $x \notin g$ for every $(x, 0) \in S$. $\diamond$

An algorithm which returns a name of a concept that is consistent with a set of examples S is called a *fitting*, since it "fits" a concept to the given examples. As always, we want a fitting to work efficiently. The running time of the fitting should be bounded by a polynomial in two variables. The first is the *length* of S, which we define as the sum of the lengths of the various $x \in X$ that S contains. The second is the size of the shortest consistent concept. For this, we will extend the l_{min} notation as follows. If S is a set of examples, then $l_{min}(S, R)$ is the size of the concept $f \in \mathcal{F}$ with smallest size that is consistent with S. If no such consistent $f \in \mathcal{F}$ exists, then $l_{min}(S, R) = \infty$.

Definition 1.14 An algorithm Q is said to be a *fitting* for a concept class $\mathcal{F}$ in representation R if

1. Q takes as input a set S of examples.
2. If there exists a concept in $\mathcal{F}$ that is consistent with S, then Q outputs a name of such a concept.

If Q is a deterministic algorithm such that the number of computational steps of Q is bounded from above by a polynomial in the length of S and $l_{min}(S, R)$, then Q is called a *polynomial-time fitting*. $\diamond$

As the next theorem (Theorem 3.1 of [Nat91]) shows, the existence of such a fitting is indeed sufficient for the polynomial-time PAC learnability of a concept class of polynomial VC-dimension.

Theorem 1.2 *Let $\mathcal{F}$ be a concept class of polynomial VC-dimension, and R be a representation of $\mathcal{F}$. If there exists a polynomial-time fitting for $\mathcal{F}$ in R, then $\mathcal{F}$ is polynomial-time PAC learnable in R.*

Conversely, it is also possible to give a necessary condition for polynomial-time PAC learnability in terms of so-called *randomized* polynomial-time fittings. We will not go into that here (see Theorem 3.2 of [Nat91]), but just mention that it can be used to establish negative results: if no such fitting for $\mathcal{F}$ in R exists, $\mathcal{F}$ is not polynomial-time PAC learnable in R.

Example 1.2 Consider an infinite sequence of properties $p_1, p_2, \ldots$ For concreteness, suppose the first properties of this sequence are the following:

p_1: "is a mammal"
p_2: "is green"
p_3: "is grey"
p_4: "is large"
p_5: "is small"
p_6: "has a trunk"
p_7: "smells awful"
...

Let us identify an animal with the set of its properties. Then we can roughly represent an animal (that is, an individual animal, not a species) by a finite binary string, i.e., a finite sequence of 0s and 1s, where the ith bit is 1 iff the animal has property p_i. Here we assume that a binary string of length n tells us whether the animal does or does not have the properties $p_1, \ldots, p_n$, while it tells us nothing about the further properties $p_{n+1}, p_{n+2}, \ldots$ Thus, for instance, some particular small, green, awfully smelling, trunk-less mammal could be represented by the string 1100101. Note that not every binary string can represent an animal. For instance, 111 would be an (impossible) mammal which is both green and grey at the same time. Similarly, since an animal cannot be large and small at the same time, a binary string cannot have 1 at both the 4th and the 5th bit.

Let us suppose our domain X is a set of binary strings, each of which represents some particular animal. Then, simplifying matters somewhat, we can identify a species with the set of animals that have the "essential" characteristics of that species. Thus, for instance, the concept 'elephant' would be the set of all large, grey, mammals with a trunk: all strings in X that have (possibly among others) properties p_1, p_3, p_4 and p_6.

How could an algorithm learn the target concept 'elephant'? Well, it would receive positive and negative examples for this concept: strings from X together with a label indicating whether the animal represented by the string is an elephant or not. Hopefully, it would find out after a number of examples that a string is an elephant iff it has (possibly among others) properties p_1, p_3, p_4 and p_6. Consider the following representation: a conjunction of p_i's is a name of the concept consisting of all strings which have (possibly among others) the properties in the conjunction. Then the conjunction $p_1 \wedge p_3 \wedge p_4 \wedge p_6$ would be a name of the concept 'elephant', and the learning algorithm could output this conjunction as a name of the concept it has learned.

It turns out that the concept class that consists of concepts representable by a conjunction of properties is polynomial-time PAC learnable (see Example 2.5 of [Nat91]). Thus, if $\mathcal{F}$ is a concept class, each member of which is a species of animals that can be represented by some finite conjunction of properties, then a polynomially-bounded number of examples and a polynomially-bounded number of steps suffices to learn some target concept (species) approximately correctly. In fact, much more complex concept classes are polynomial-time PAC learnable as well. For an overview of positive and negative results, see [Nat91, AB92, KV94]. ◁

1.7 Some Related Settings

The standard PAC setting of the previous sections may be varied somewhat. In this section, we will mention some alternatives.

1.7.1 Polynomial-Time PAC Predictability

In the ordinary PAC setting, a PAC algorithm for a concept class $\mathcal{F}$ reads examples from an unknown target concept f from $\mathcal{F}$, and has to construct a concept g, *also from* $\mathcal{F}$, which is approximately correct. This may lead to a seemingly paradoxical situation: we would expect that learning a superset of $\mathcal{F}$ is at least as hard as learning $\mathcal{F}$ itself, but this need not be the case in the ordinary PAC setting. Namely, it may be that there is no polynomial-time PAC algorithm for some concept class $\mathcal{F}$ in some representation R, while for some larger concept class $\mathcal{G} \supset \mathcal{F}$ there *is* such a polynomial-time PAC algorithm. The latter algorithm, when given examples for some target concept $f \in \mathcal{F}$, always constructs a name of a probably approximately correct concept $g \in \mathcal{G}$ in polynomial time. Still, $\mathcal{F}$ itself may be hard to learn, because the requirement that the output concept should be a member of $\mathcal{F}$ may be very hard to meet.

We can take this into account by loosening the requirement on g somewhat, and allow it to be a member of a broader concept class $\mathcal{G}$, of which $\mathcal{F}$ is a subset. This gives the learning algorithm more freedom, which may facilitate the learning task. Suppose we have a concept class $\mathcal{F}$, a broader concept class $\mathcal{G} \supseteq \mathcal{F}$, and a representation R of $\mathcal{G}$ (which is of course also a representation of $\mathcal{F}$). Suppose, furthermore, that there exists a learning algorithm L for $\mathcal{F}$ in R, which is just like a PAC algorithm for $\mathcal{F}$ in R, except that it outputs a name of a concept g such that $g \in \mathcal{G}$ but not necessarily $g \in \mathcal{F}$. In this case, we say that L is a *PAC prediction algorithm for* $\mathcal{F}$ *in* R *in terms of* $\mathcal{G}$ and $\mathcal{F}$ is *PAC predictable in* R *in terms of* $\mathcal{G}$. If, furthermore, the time complexity of algorithm L is bounded by a polynomial in $1/\varepsilon$, $1/\delta$, n, and l, we say that $\mathcal{F}$ is *polynomial-time* PAC predictable in R in terms of $\mathcal{G}$. If some $\mathcal{G}$ exists such that $\mathcal{F}$ is polynomial-time PAC predictable in R in terms of $\mathcal{G}$, we will simply say that $\mathcal{F}$ is polynomial-time PAC predictable in R.

Clearly, if some concept class $\mathcal{F}$ is polynomial-time PAC learnable in some R, it is also polynomial-time PAC predictable in R: simply put $\mathcal{G} = \mathcal{F}$. Hence the setting of polynomial-time PAC predictability may be used to establish negative results: if we can prove that some concept class $\mathcal{F}$ is not polynomial-time PAC predictable in R in terms of any $\mathcal{G}$, we have thereby also shown that $\mathcal{F}$—as well as any superset of $\mathcal{F}$—is not polynomial-time PAC learnable in R. The converse need not hold: some classes are polynomial-time PAC predictable, but not polynomial-time PAC learnable (see Sections 1.4 and 1.5 of [KV94] for an example). Hence polynomial-time PAC predictability is strictly weaker than polynomial-time PAC learnability.

1.7.2 Membership Queries

We may facilitate the learning task by allowing a PAC algorithm to make use of various kinds of *oracles*. An oracle is a device which returns answers to certain questions, which are called *queries*. For the PAC algorithm that uses an oracle, the oracle is like a black box: you pose a question and get an answer, but do not know *how* the oracle constructs it answer. Like the EXAMPLE procedure, oracles are assumed to run in at most some fixed constant number of steps.

The most straightforward kind are the *membership* queries. Here the oracle takes some $x \in X$ as input, and returns 'yes' if x is a member of the target concept, and 'no' if not. Clearly, the oracle somehow has to have knowledge about the domain. Two justifications for assuming an oracle can be given:

1. Induction can be compared with a simplified picture of the work of a scientist. Consider a particle physicist. The physicist may not know the general laws that characterize the objects in his domain of inquiry, but he can obtain knowledge about certain specific instances of those concepts by doing experiments. Posing a membership query to an oracle is similar to doing an experiment in science, which is like "posing a question to nature".

2. When learning, a student may have a teacher who can answer questions about whether some object has a certain property or not. It need not be the case here that the student only learns what the teacher already knows. We only assume the teacher has sufficient knowledge of individual objects of the concepts. The teacher may know all about the particular instances of the concepts, and yet be pleasantly surprised by the concept that a smart student comes up with. Translating this analogy to induction, the oracle acts as the teacher, while the learning algorithm is the student.

If a concept class $\mathcal{F}$ is polynomial-time PAC learnable in some R by an algorithm which makes membership queries, we will say that $\mathcal{F}$ is polynomial-time PAC learnable in R *with membership queries*. Analogously, we can define PAC predictability with membership queries. Note that if an algorithm makes membership queries, it in a way "creates its own examples." Note also that a polynomial-time algorithm can make at most a polynomial number of queries, since each query counts for at least one computational step.

Equivalence queries need a more fancy oracle, which is discussed in the next subsection. For an overview of other kinds of queries, we refer to [Ang88].

1.7.3 Identification from Equivalence Queries

While polynomial-time PAC predictability is strictly weaker than polynomial-time PAC learnability, *polynomial-time identification from equivalence queries*, introduced by Angluin [Ang87b], is strictly stronger. In this setting, we have an oracle which takes a name of a concept g as input, and answers 'yes' if g equals the target concept f, and 'no' otherwise. In case of a 'no', it also returns a randomly chosen *counterexample* $x \in f \Delta g$. There is no need for the oracle to provide the correct label of the counterexample x, because the algorithm can

find this out for itself: if $x \in g$ then $x \notin f$, and if $x \notin g$ then $x \in f$. When equivalence queries are available, the requirement that an algorithm outputs a name of an approximately correct concept is replaced by the requirement that the target concept is *identified exactly*: an algorithm that is allowed to make equivalence queries should output a name of the target concept.

Consider a concept class $\mathcal{F}$ and a representation R of $\mathcal{F}$. Let L be an algorithm which uses equivalence queries in order to learn some unknown concept $f \in \mathcal{F}$ under some unknown probability distribution $\mathbf{P}$, and which takes as input an upper bound l on $l_{min}(f, R)$ and an upper bound n on the length of the counterexamples from the oracle. If this algorithm always outputs a name of the target concept, we say $\mathcal{F}$ is *identifiable from equivalence queries* in R. If the running time of the algorithm L is bounded by a polynomial in its inputs l and n, then $\mathcal{F}$ is *polynomial-time* identifiable from equivalence queries in R. As in the case of membership queries, an algorithm with a polynomially-bounded running time can make only a polynomially-bounded number of equivalence queries.

It is shown in Section 2.4 of [Ang88] that if a concept class is polynomial-time identifiable from equivalence queries in some R, then it is also polynomial-time PAC learnable in R. The converse does not hold. Thus, while PAC predictability can be used to establish negative results, identification from equivalence queries may be used for positive results: if we can prove that some concept class $\mathcal{F}$ is polynomial-time identifiable from equivalence queries, we have thereby also shown that $\mathcal{F}$, as well as any subset of $\mathcal{F}$, is polynomial-time PAC learnable in R.

We may also allow an algorithm to make both equivalence queries and membership queries. Angluin [Ang87b] calls this combination a "minimally adequate teacher": if a teacher wants to teach some target concept to his student, he should be able to answer student's questions about whether some object is in the target concept (membership queries), and he should be able to judge whether some concept the student comes up with, is really the target concept, and give a counterexample if not (equivalence queries). If polynomial-time identification of $\mathcal{F}$ from equivalence queries is done by an algorithm which makes use of equivalence queries as well as membership queries, then we say $\mathcal{F}$ is *polynomial-time identifiable from equivalence and membership queries* in R. This implies polynomial-time PAC learnability with membership queries.

1.7.4 Learning with Noise

In many real-world learning tasks, examples may contain errors (*noise*), which may for instance be due to inaccurate measurements. There are various ways in which the analysis of noise may be modelled in the theoretical setting for PAC learnability. We will discuss only two kinds of noise here: Valiant's *malicious* noise [Val85], also sometimes called *adversarial* noise, and Angluin and Laird's *random classification* noise [AL88]. For other kinds of noise, see [Lai88, Slo95].

Firstly, in the malicious noise model, a malicious adversary of the learning algorithm tinkers with the examples: for each example that the learning algorithm reads, there is a fixed, unknown probability $0 \leq \eta$ that the adversary has

changed the original, correct example (x, y) to any other (x', y')-pair he chooses. Since y' may not be the correct label for x', the adversary may introduce noise in this way. The adversary is assumed to be omnipotent and omniscient—in particular, he has knowledge of the learning algorithm he is trying to deceive. This means that the learning algorithm should be able to cope even with the worst possible changes in the examples.

Secondly, in the random classification noise model, the EXAMPLE procedure is replaced by a procedure EXAMPLE^{η}, and there is a fixed, unknown probability $0 \leq \eta < 0.5$ that the label of an example provided by this procedure is incorrect. For instance, suppose $\eta = 0.1$. If a learning algorithm receives an example (x, y) from EXAMPLE^{η}, then there is a probability of 10% that y is incorrect.

In both models, the actual noise rate η is unknown to the learning algorithm. However, an upper bound η_b on the noise rate is given as an additional input parameter to a PAC algorithm, where $0 \leq \eta \leq \eta_b < 0.5$. This η_b is added as a parameter to the time complexity function as well. If there is a PAC algorithm for a concept class $\mathcal{F}$ in some representation R, working in the presence of malicious (resp. random classification) noise, with time complexity bounded by a polynomial in $1/\varepsilon$, $1/\delta$, n, l, and $1/(1-2\eta_b)$, then $\mathcal{F}$ is said to be polynomial-time PAC learnable in R *with malicious (resp. random classification) noise*. Similarly, we can define PAC predictability with malicious or random classification noise.

1.8 Summary

This thesis is concerned with *computational learning theory*: the study of algorithmic ways to learn from examples. The dominant formal model of learnability in Artificial Intelligence is Valiant's model of *approximately correct learning*. In this model, a *concept* is simply a subset of a domain X, and a *concept class* is a set of concepts. A *PAC algorithm* reads examples for an unknown *target* concept (taken from some concept class), drawn according to an unknown probability distribution, and learns, with tunably high probability, a tunably good approximation of the target concept. A concept class $\mathcal{F}$ is *polynomial-sample PAC learnable* if a PAC algorithm exists for $\mathcal{F}$ that uses only a polynomially-bounded number of examples, and is *polynomial-time PAC learnable* if the algorithm uses only a polynomially-bounded number of steps. In the latter case, the algorithm should output a *name* of the learned concept in some polynomially evaluable *representation*. An oracle for *membership queries* can inform the learner whether a specific object is in the target or not. Polynomial-time *PAC predictability* is weaker than polynomial-time PAC learnability, while polynomial-time *identification from equivalence queries* is stronger. When *noise* is involved, the examples may sometimes be incorrect.

Chapter 2

Application to Language Learning

2.1 Introduction

In the course of the 20th century, *language* has become the focal interest of philosophy. Many of the central philosophical problems—logical, epistemological, anthropological, ethical, and even metaphysical—are bound up with the intricacies of human language. Investigating *how* human beings learn languages, and which languages *can* be learned by human beings may tell us a lot about those intricacies, and should therefore be of great importance to philosophy.

Seeking knowledge about language, where better to turn than to *linguistics*, the science of language? And within linguistics, whom better to turn to than Noam Chomsky, the man who made linguistics the most scientific of all humanities? One of Chomsky's most interesting claims concerns the *innateness* of important aspects of our natural languages. For this claim, he has been severely attacked by various empiricist philosophers, among them Putnam and Quine. While Chomsky argues that children can only acquire language in virtue of having specific innate language acquisition mechanisms, Putnam and Quine argue that general (not language-specific) learning mechanisms may suffice for language acquisition.

This chapter is an attempt to settle this issue in Chomsky's favour by means of a mathematical argument: we will use results from computational learning theory to establish that children would not be able to learn their native language if they started without any pre-knowledge of the language they have to learn. In other words, we provide a formal "proof" that *general* learning mechanisms cannot explain why children acquire language as successfully as they in fact do: language acquisition must be based on certain propensities and biases which direct the child towards certain kinds of languages and away from others. Where do these biases come from? The most plausible answer is that they are innate.

The chapter is organized as follows. We start by sketching some background concerning Chomskyan linguistics and the innateness of language. In order to be able to state formal results about language learnability, we need to provide two things: a formal model of learning, and a formal model of languages

and grammar. The first has been dealt with in the previous chapter, while Section 2.3 provides formal counterparts to the notions of a language and a grammar. One of the main self-imposed goals of philosophers is to bring out presuppositions. In Section 2.4, we follow this laudable practice, making explicit the main assumptions and presuppositions of our analysis of language learnability. Sections 2.5 to 2.11 form the main part of the chapter. Here we show that the set of languages a child can learn must be severely constrained in order to enable efficient learning to take place. Finally, in Section 2.12 we extrapolate this argument to other kinds of learning.

2.2 A Brief History of the Chomskyan Revolutions

In this section we will give a brief and incomplete overview of the revolution caused in linguistics by the work of Noam Chomsky.[1] Actually, we may distinguish between two revolutions: the first replaced the behavioristic paradigm by the paradigm of transformational-generative grammar; the second involved important changes in the transformational-generative framework, yielding Chomsky's current principles-and-parameters framework.

2.2.1 Against Behaviorism

Linguistics BC, Before Chomsky, was dominated by behaviorism. Accordingly, a good place to start our story is Chomsky's review of *Verbal Behavior*, a book by the leading behaviorist B. F. Skinner. In his book, Skinner attempted to extend the behaviorist approach to the study of language use by human beings. The behaviorist picture of science amounts to the following: you have some object, for instance an animal, which reacts or responds in certain ways to certain stimuli from the environment. The task of the scientist is to find laws which describe the relations between stimulus and response, on the basis of experiments where you vary the stimulus and observe how the response changes. Previously, that approach had mainly been restricted to very small contexts, for instance rats in mazes, where behaviorist concepts like "stimulus", "response", "reinforcement" could be precisely defined by reference to simple measuring apparatus.

Chomsky's critique of Skinner's book was simple, yet effective: the extrapolation of the behaviorist approach to the area of human language use leaves the key behaviorist concepts empty. The problem for the behaviorist is: *what are the stimulus and the response in case of linguistic behavior?* In order to give results, the behaviorist approach requires a precise definition of things like stimulus and response, as well as the ability to somehow measure or determine those things. In a simple laboratory experiment with animals, this can indeed be done. However, if we supplant the laboratory terminology to the much more complex case of language learning, it either becomes non-applicable (if we take the terminology literally) or empty (if we take it metaphorically). Skinner's

[1] This summary is mainly based on a number of recent linguistic texts [Bot89, New91, Har93, Pin94], to which we refer the reader for more detail.

attempt to extend the behaviorist approach to human language use failed, and so have later attempts. In fact, it seems plausible that the precise definitions and ways of measuring that the behaviorist requires, are simply unavailable in the complex area of linguistic behavior.

2.2.2 Transformational-Generative Grammar

If language use and learning cannot be described behavioristically, then what? A few years before his Skinner-review, Chomsky had himself put forward a radically different linguistic theory. He distinguishes between linguistic *performance* and *competence*. Performance is the way a person actually or potentially *uses* language; competence is what he, perhaps unconsciously, *knows* about that language. The distinction is crucial: natural languages contain an infinite set of sentences which have *never* been used before (and hence are not amenable to behavioristic analysis), yet which would easily be recognized as grammatical by any competent native speaker. Because performance varies too much with the contingencies of context, it is not well suited for scientific inquiry; competence is the appropriate target for linguistics. Thus we need a model which specifies the knowledge native speakers have of the set of all syntactically correct sentences, rather than the ones that have actually been used or uttered.

Throughout his career, Chomsky's linguistic research has been motivated by the following problem: what makes it possible that almost all children acquire near-perfect competence of their native language, despite the poverty of the stimulus they receive? When children learn their first language, the only "input" they receive are the sentences they hear from their parents and others. This set of sentences does not uniquely determine a language: many different languages are consistent with the input the child receives. Nevertheless, it is an empirical fact that children all fill in the gaps in more or less the same way, learning approximately the same language. From this Chomsky concludes that children must be born with a strong linguistic bias, consisting of constraints on the set of possible languages. These constraints, *Universal Grammar*, lead children to learn only very specific languages from the input they receive, ignoring the infinite number of other languages compatible with the input.

Universal Grammar is incarnated in what Chomsky calls the *language faculty*[2], and what others sometimes call the *language organ* or the *language instinct*, which is supposed to be a more or less separate module in the mind/brain. (Chomsky often uses the term "mind/brain" in order to forestall discussions of the "dualism vs. monism" type.) In Chomsky's view, the main task of linguistics is to investigate the properties of Universal Grammar. Thus linguistics would give us information about the workings of our mind/brain. In fact, Chomsky has stated at several places that he is mainly interested in linguistics not for its own sake, but because it is a way to gain knowledge about the mind [Har93, p. 11]. This is in sharp contrast to the earlier behaviorist approach, which eschewed anything mental.

[2]The term Universal Grammar is used with systematic ambiguity, referring both to the initial, innate state of the language faculty at birth, and to the properties shared by all natural languages. For the latter, see [Haw88].

Chomsky's initial broad model of language competence, first described in his groundbreaking work [Cho57] and elaborated in more painstaking detail in [Cho65], roughly amounts to the following. Having competence of some language amounts to "having", in some sense, a *transformational-generative* grammar for that language "in your head". A transformational-generative grammar for a language determines the set of syntactically correct sentences of that language. It consists of two parts: a *base* part and a *transformational* part. The first part generates *deep* structures of sentences, the second part transforms these into the *surface* structures that we normally would call sentences. Further operations on deep structure were to yield the *interpreted form* (or meaning) of a sentence, while further operations on surface structures would yield the *phonological* form of a sentence. However, in this chapter we will ignore such further linguistic issues, restricting attention to syntax.

The base part is a *generative grammar*, which consists of a set of *phrase structure rules* and a *lexicon*. The phrase structure rules recursively specify the forms a sentence may have. For example, such rules might state "a sentence can consist of a noun phrase followed by a verb phrase" and "a noun phrase can consist of a determiner followed by a noun." The lexicon is like a dictionary: it contains the words that may be plugged into those forms. One entry might for instance be "*dog*: singular animate noun". Inserting words into a sentence form yields a deep structure. The base part of a grammar thus generates a set of deep structures. An example of such a deep structure might be the following tree, which gives a structural description of the sentence "The dog bites the man."

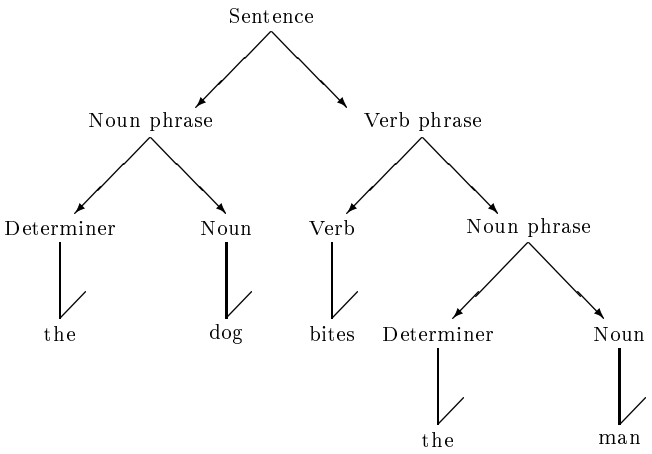

Figure 2.1: Deep structure of the sentence "The dog bites the man"

The transformations that make up the second part of a transformational-generative grammar map deep structures to surface structures. For example, two transformations may take the above deep structure into the following surface structures, respectively:

1. The dog bites the man. (the identity transformation)
2. The man was bitten by the dog. (a "passivizing" transformation)

The relationship between sentences 1 and 2 is brought out by the way they have been constructed: both stem from the same deep structure.

The base part of a transformational-generative grammar—phrase structure rules plus lexicon—was intended to be fairly restricted in power, hopefully only context-free (see Section 2.3 for the definition of this term). Unfortunately, the transformational part of a grammar is potentially too expressive: it can be shown that any grammatically describable language can be generated by means of a transformational-generative grammar [PR73, BC74], even if we choose a very simple, fixed base part. This means that the transformational-generative framework by itself, including the choice of the base part, does not seem to make any substantive claims about Universal Grammar. Accordingly, claims about Universal Grammar had to be phrased in terms of *constraints* on the allowed base part and, particularly, transformations. Eventually, the problems in formulating adequate constraints led up to a second revolution in linguistics.

2.2.3 Principles-and-Parameters

Chomsky's current *principles-and-parameters* model does away with phrase structure rules altogether, and with most of the transformations as well [Cho81, Cho86].[3] Instead, much more emphasis is placed on the information in the lexicon than in the earlier model. Sentences are generated directly from the lexicon by means of the interaction of a number of subsystems, each consisting of various general principles. In this new model, the principles are innate—and hence the same for all natural languages—except for some parameterized variation. Thus innate Universal Grammar specifies "schemes" of principles, which have certain open parameters, and it specifies the range of possible values those parameters may take. Fixing the parameters in the principle schemes yields the principles that govern particular natural languages. Therefore, apart from the lexicon, each natural language can be characterized by the particular values of the parameters for that language. Accordingly, for a child, learning a language now amounts to two things: (1) determining the particular values of the parameters that yield the principles of its native language, and (2) acquiring the lexicon.

2.2.4 The Innateness of Universal Grammar

The above pages briefly mentioned the philosophically most interesting of Chomsky's claims: the *innateness* of Universal Grammar. Innateness has of course been a topic for philosophical discussion for years. Particularly in the 17th and 18th century the debate was on. The "rationalists" Descartes and Leibniz are commonly considered to be on the innate side, the "empiricists" Locke and Hume on the other. Thus for instance Descartes (as cited on p. 48 of [Cho65]) takes the ideas of figures, pain, colour and sound to be innate, while Book I of

[3] It should be (foot)noted that nowadays Chomsky's work is much less dominant than it was in, say, the 1960s. In present-day linguistics, Chomsky's approach is one among several alternative approaches. Other are based on, for instance, neural networks or various kinds of constraints (see [Sei97, PS97] and the references therein).

Locke's *Essay* [Loc93] argues against all innate notions and principles. As far as the Anglo-American philosophical world is concerned, it seems fair to say that the empiricist side of the discussion has been dominant. It has long been the guiding principle of empiricist philosophy that most (possibly all) knowledge derives from the senses—a principle which would be seriously undermined if important aspects of knowledge of language turned out to be innate.

Chomsky's work has revived the old debate on innateness in an updated guise: the discussion has shifted from innate *ideas* to innate *mechanisms*. As far as language learning is concerned, Chomsky explicitly sides with the rationalists:

> "In general, then, it seems to me correct to say that empiricist theories about language acquisition are refutable wherever they are clear, and that further empiricist speculations have been quite empty and uninformative. On the other hand, the rationalist approach exemplified by recent work in the theory of transformational grammar seems to have proved fairly productive, to be fully in accord with what is known about language, and to offer at least some hope of providing a hypothesis about the intrinsic structure of a language-acquisition system that will meet the condition of adequacy-in-principle and do so in a sufficiently narrow and interesting way so that the question of feasibility can, for the first time, be seriously raised." [Cho65, pp. 54–55]

Thus it is not surprising to find that he has been attacked by various contemporary philosophers of a more empiricist bent, such as Putnam and Quine.[4] Unfortunately, the discussion has been hampered by mutual misunderstandings (see for instance [Cho75, Qui75][5].) Both positions are not as extreme as their respective opponents sometimes take them to be. As Rosemont [Ros78] notes, the Chomskyans certainly acknowledge that a child develops language from experiental data (though they would perhaps prefer to say that language *grows* in a child, triggered by experience, rather than that is being *learned* from experience). On the other hand, most present-day empiricists have watered down their empiricism somewhat, and would agree that we are not born as a *tabula rasa*, but have various innate capacities and biases. An example of the latter would be a general innate measure of "similarity" used for induction [Qui69].

Perhaps the best way to state the debate is as follows. While Quine and other empiricists are willing to admit innate biases and learning mechanisms, these are *general-purpose* mechanisms. Chomsky, on the other hand, postulates innate mechanisms which are *specific for language*. In other words, while empiricists still hang on to the idea that children acquire language by means of the same general-purpose learning mechanisms that enable them to learn, e.g.,

[4] Putnam certainly was an empiricist at the time he wrote his critique of Chomsky [Put71, Put83], but seems harder to classify now. Quine is still as empiricist as he ever was.

[5] However, it is interesting to note that of the fifteen replies that Quine wrote in [DH75], the one to Chomsky is the longest and most detailed. This suggests that Quine took Chomsky's criticisms more seriously than those made by other philosophers.

to recognize faces or to wash their hands after dinner, Chomsky dispelled with this idea.

In the remainder of this chapter, we will see that results from computational learning theory indicate that Chomsky is in the right here. In particular, some pre-knowledge of the language that is to be learned is required in order to enable efficient language acquisition, and this pre-knowledge is most likely innate. It is perhaps surprising that this conclusion can be established by means of a *mathematical* argument. To be sure, Chomsky himself writes the following about the innateness of particular linguistic mechanisms:

> "You can't demonstratively prove it is innate—that is because we are dealing with science and not mathematics; even if you look at the genes you couldn't prove that. In science you don't have demonstrative inferences; in science you can accumulate evidence that makes certain hypotheses seem reasonable, and that is all you can do—otherwise you are doing mathematics."
> [Cho83, p. 80], as cited in [Bot89, p. 199].

Indeed, we cannot formally prove that certain *specific* grammatical principles must be innate. Still, what we *can* prove—given the presuppositions outlined in Section 2.4—is that *some* bias must be present in order to make language learning feasible, and this bias is probably innate, since there is no plausible alternative source where it might have come from.

2.3 Formalizing Languages and Grammars

Since the aim of this chapter is to give a formal "proof" of the necessity of innate grammatical biases in language learning, we need to formalize the notions of language and grammar. This is the job of the next three subsections.

2.3.1 Formal Languages

Let us consider some fixed alphabet Σ, for instance the set consisting of the 26 letters 'a',...,'z' (in lower as well as upper case), the 10 digits '0',...,'9', and some interpunction symbols like '.', ':', '?', and blank space. For any set of symbols S, we use S^* to denote the set of all finite strings (concatenations) of symbols from S, and S^+ to denote the set of all finite *non-empty* strings. A *sentence* will simply be a member of Σ^+, i.e., a finite non-empty string of symbols from Σ.

We can only speak of grammatically correct or incorrect sentences relative to a *language*. We will take a language to be simply a (possibly infinite) set L of sentences: any subset of Σ^+ is a language. A sentence s is grammatically correct for L if $s \in L$, and grammatically incorrect otherwise.[6] Thus, for instance, the English language is simply the set E of all English sentences. The sentence "The dog ate my homework" would be grammatically correct in English (it would be

[6] This is similar to making grammatical correctness relative to a grammar: given a grammar G for a language L, a sentence s is grammatically correct iff G generates s.

a member of E), while the sentences "dog the blab" and "De hond heeft mijn huiswerk opgegeten" would not. Of course, for natural languages the boundaries between grammatically correct and incorrect sentences are not that sharp. In fact, Chomsky [Cho65] has suggested that grammaticality may be a matter of degree, though he has not added much subsequent flesh to this suggestion. In this chapter, we will assume grammaticality to be a sharp boundary.

2.3.2 Formal Grammars

A common way to specify a language is by giving its *grammar*. We will define a grammar as a set of rules, called *productions*. Using productions, a grammar generates sentences starting from an initial symbol **S** (for 'sentence'). Let N be a finite set of symbols called *non-terminals*. N should at least contain the symbol **S**. The symbols in the alphabet Σ are called *terminals*, and we will assume Σ and N to be disjoint. In order to distinguish typographically non-terminals from terminals, we will write down non-terminals in a bold facetype.

A *production* is something of the form $\mathcal{A} \to \mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are finite strings over $\Sigma \cup N$, and $\mathcal{A}$ contains at least one non-terminal. $\mathcal{A}$ will be referred to as the *left-hand side* of the production, and $\mathcal{B}$ as the *right-hand side*. A *grammar* G is a finite set of productions, such that at least one of the productions in G has **S** as left-hand side.[7]

How does a grammar relate to a language? The productions in a grammar G function as *rewriting* rules, which allow you to replace, in some string, the left-hand side of a production by the right-hand side of that production. G can generate a sentence (a string of terminals) as follows:

1. Start with the string $\mathcal{S} = \mathbf{S}$.
2. Repeat the following:
    1. Find a production $\mathcal{A} \to \mathcal{B} \in G$ such that $\mathcal{A}$ occurs somewhere in the string $\mathcal{S}$.
    2. Apply the production: replace one occurrence of $\mathcal{A}$ in $\mathcal{S}$ by $\mathcal{B}$.

    until $\mathcal{S}$ is a sentence (i.e., contains only terminals).

The language generated by a grammar G, denoted by $L(G)$, is the set of all sentences which can be generated in this way.

An example will make this clearer. Consider a set N consisting of the three non-terminals **S** (for Sentence), **N** (for Name), and **V** (for Verb phrase). Let G be the grammar consisting of the following five productions:

1. $\mathbf{S} \to \mathbf{N}\ \mathbf{V}$
2. $\mathbf{N} \to$ John
3. $\mathbf{N} \to$ Paul

[7] A grammar is often defined more formally as a 4-tuple $G = (N, \Sigma, P, \mathbf{S})$, where N is the set of non-terminals, Σ is the set of terminals, P is the set of productions, and **S** is the starting symbol. We have simplified this to $G = P$, because our starting symbol will always be **S**, and the sets N and Σ can be read off from the set of productions.

4. **V** $\rightarrow$ hates **N**
5. **V** $\rightarrow$ thinks that **S**

We do not make a formal distinction between the phrase structure rules of a grammar and its lexicon; both are represented by means of productions. The first production states that a sentence consists of a name followed by a verb phrase. The names can be either 'John' or 'Paul'. The last two productions show how the verb phrase **V** can be expanded. Note that the second of these two productions introduces the non-terminal **S** again. This grammar can for instance generate the sentence "John hates Paul" as follows:

$$\mathbf{S} \overset{1}{\Rightarrow} \mathbf{N}\ \mathbf{V} \overset{2}{\Rightarrow} \text{John}\ \mathbf{V} \overset{4}{\Rightarrow} \text{John hates}\ \mathbf{N} \overset{3}{\Rightarrow} \text{John hates Paul}$$

We start with **S** and apply productions until we end up with a string without non-terminals. The application of production i is denoted above by $\overset{i}{\Rightarrow}$. Such a sequence of applications of productions is called a *derivation* of the string from the grammar. Note that a derivation corresponds to the kind of structural description that is embodied in the tree on p. 22.

Similarly, G can generate the sentences "John hates John", "Paul hates John", and "Paul hates Paul". We can also use G generate the more complex sentence "Paul thinks that John hates Paul", the derivation of which is:

$$\mathbf{S} \overset{1}{\Rightarrow} \mathbf{N}\ \mathbf{V} \overset{3}{\Rightarrow} \text{Paul}\ \mathbf{V} \overset{5}{\Rightarrow} \text{Paul thinks that}\ \mathbf{S} \overset{1}{\Rightarrow} \text{Paul thinks that}\ \mathbf{N}\ \mathbf{V} \overset{4}{\Rightarrow} \text{Paul thinks that}\ \mathbf{N}\ \text{hates}\ \mathbf{N} \overset{2}{\Rightarrow} \text{Paul thinks that John hates}\ \mathbf{N} \overset{3}{\Rightarrow} \text{Paul thinks that John hates Paul}$$

$L(G)$, the language generated by G, is the set of all sentences which can be generated in this way. Note that $L(G)$ is infinite, due to the reintroduction of **S** in the fifth production: the language contains the sentences "John thinks that Paul hates John", "John thinks that Paul thinks that John hates Paul", etc. It should be clear that the language generated in this way is only a tiny subset of a natural language (e.g., English). On the other hand, the example shows that small—and yet infinite!—fragments of language can already be generated with very simple grammars. This holds out the hope that larger parts of natural languages can be generated by larger grammars, and perhaps it is even possible to give a complete grammar for a natural language.

2.3.3 The Chomsky Hierarchy

Clearly, some languages are more complex than others. The complexity of a language is related to the complexity of the simplest grammar which generates that language. Below we define the *Chomsky hierarchy*, consisting of the classes of Type 3, Type 2, Type 1, and Type 0 languages, with increasing grammatical complexity.

- A *Type 3* (or *regular*) grammar contains only productions in which the left-hand side is a non-terminal, and the right-hand side is either a terminal or the concatenation of a terminal and a non-terminal.

- A *Type 2* (or *context-free*) grammar contains only productions in which the left-hand side is a non-terminal, while the right-hand side is an arbitrary non-empty string of terminals and non-terminals.
- A *Type 1* (or *context-sensitive*) grammar contains only productions in which the right-hand side is at least as long as the left-hand side.[8]
- A *Type 0* grammar may contain any kind of productions.

A language is of Type i ($i = 0, 1, 2, 3$) if it can be generated by a grammar of Type i. For instance, the John/Paul-grammar is a Type 2 (context-free) grammar, and hence generates a Type 2 (context-free) language. Of what Type are full natural languages? This is a question to which we will return later.

We will now informally state some important results from the theory of formal languages (for technical details and proofs, see [HU79]). Firstly, it can be shown that the class of Type 3 languages is a proper subset of the class of Type 2 languages. Similarly, Type 2 is a proper subset of Type 1 and Type 1 is a proper subset of Type 0. Type 1 languages are *recursive*: there exists an algorithm for deciding whether a given sentence is a member of the language generated by a given Type 1 grammar. Type 0 languages are *recursively enumerable*: there exists an algorithm which enumerates the (possibly infinite) set of sentences in the language generated by a given Type 0 grammar. Membership of a sentence in a language is semi-decidable, but not always decidable for a given Type 0 language. Type 0 grammars are equivalent to Turing machines, in the sense that a language is of Type 0 iff it is accepted by some Turing machine. Finally, even though the class of Type 0 languages is the broadest class in the hierarchy, it still does not comprise *all* possible languages: some languages are not Type 0 languages.[9]

2.4 Simplifying Assumptions for the Formal Analysis

In the next sections, we will give a formal analysis of language learning from example sentences in the PAC setting. The main objective there is to show that unbiased language learning is just too hard—and hence cannot be what children actually do. From this it would follow that children *must* have some biases which influence the language they learn and the way they learn it.

It will be clear to all but the most naive readers that the real world is simply too big to model completely. Inevitably, a formal analysis involves a number of

[8] A context-sensitive grammar may equivalently be defined as a set of productions of the form $\mathbf{A} \rightarrow \mathcal{B}/\alpha\_\_\beta$. Here $\mathbf{A} \rightarrow \mathcal{B}$ is simply a context-free production which, however, may only be applied as a rewriting rule in case $\mathbf{A}$ is surrounded by α on the left and β on the right. That is, $\alpha\_\_\beta$ specifies the *context* in which the rule may be applied (α and/or β may be empty).

[9] A very quick proof of this: (1) Σ^+, the set of all sentences, is denumerably infinite; (2) the set of all languages is the power set of Σ^+ (i.e., the set of all sets of sentences), and is therefore uncountable; (3) the set of all grammars is only denumerably infinite; (4) thus there are more languages than there are grammars, which implies (5) that some languages cannot be generated by any grammar, and hence are not Type 0 languages.

simplifications, which have the disadvantage of making the analysis less "realistic" and correspondingly less plausible. On the other hand, simplifying away a number of the contingencies and noise of the real world may bring out more clearly what really matters for language learning. Let us state and defend right at the outset the main simplifying assumptions we will make here.

2.4.1 Language Learning Is Algorithmic Grammar Learning

A first assumption is that the process by which a child learns a language can be described by an *algorithm*. This algorithm takes sentences from some *target language* (i.e., what is to become the child's native language) as input and learns a *grammar* for those sentences. The sources of those input sentences may be very diverse: parental speech, dialogue from television series, and so on. Actually, two distinct assumptions are at work here: (1) that language learning is *algorithmic*, and (2) that it involves learning a *grammar*. We will clarify these two assumptions separately.

Firstly, the algorithmic aspect. To say that a child's learning can be *described* by an algorithm does not imply that children consciously enact the steps of some algorithm when they are learning. It does, however, mean that there is an algorithm which, whenever it is given the same input as the child, learns the same grammar. In other words, there should be an algorithm whose input-output behaviour is *equivalent* (or, if you will, *isomorphic*) to the child's.

With this assumption, we are squarely within the tradition of cognitive science. Here *all* intelligent behaviour (which includes learning) is taken to be describable as some form of algorithmic information processing. A very fundamental theoretical justification for this may be found in the *Church-Turing thesis*. Informally, this thesis says that everything that can be accomplished by certain systematic means, whatever these may be, can also be accomplished by an algorithm as implemented in a Turing machine (see Chapter 17 of [Hof79] for more on this). If the Church-Turing thesis holds, then it seems that the process of language acquisition should indeed be describable by an algorithm.

Secondly, what does it mean to say that children learn a *grammar*? It is a well known fact that people are usually not able to state explicitly the grammar of their native language: they *follow* the rules of that grammar without consciously *knowing* those rules. That is, people may have "know-how" knowledge of language (i.e., they know how to use language), without having "know that" knowledge. Thus we have to be a bit careful when we say that children acquire a grammar. In the sequel, we will say that a child has learned a grammar G if the child, by and large, consistently *follows* that grammar: it only utters (or writes) sentences from $L(G)$. In other words, we will say that a child has learned a grammar if that grammar appropriately describes the child's linguistic behaviour—even though the child itself may be unaware of the rules of the grammar it follows. In this way a grammar provides an appropriate description of "know-how" knowledge of language.[10]

[10] Note that it is no easy matter to find out whether a child's linguistic behaviour is in accordance with some grammar G. Finding this out will usually be a matter of induction itself: if we have observed a child's linguistic utterances for quite a while, and all utterances

One further caveat has to be entered. Namely, the way we have formalized grammars (as finite sets of productions) in the previous sections is rather different from either Chomsky's transformational-generative grammar or his later principles-and-parameters model. Actually, a set of productions formalizes only the base part of a transformational-generative grammar, ignoring the transformations. Furthermore, we only focus on the set of sentences that a grammar generates, mainly ignoring the way the sentence is parsed by the grammar; that is, the only part of trees like Figure 2.1 that we are interested in, is the sentence constituted by the words at the leaves of the tree. However, any language with a transformational-generative or a principles-with-fixed-parameters grammar is a Type 0 language, and hence representable by a finite set of productions. Accordingly, restricting attention to the learnability of sets of productions does not really invalidate our analysis.

This caveat also bars invoking *semantical* considerations to argue against the present purely syntax-oriented analysis. Though a full grammar would probably let semantical issues influence the syntax, the resulting system would still be equivalent to some Turing machine (assuming the Church-Turing thesis), and hence could be redescribed in purely syntactical terms as a Type 0 grammar. Again, restricting attention to the learnability of syntax (i.e., sets of productions) does not invalidate our analysis.

2.4.2 All Children Have the Same Learning Algorithm

Furthermore, we will assume that all children can be described by the *same* learning algorithm. This is certainly a false presupposition: no two children are the same, so no doubt some children will process sentences in a different way and will learn a different grammar from the same input. Nevertheless, since the brains of children all over the world have roughly the same structure, it does seem fair to say that they probably have *approximately* the same mechanisms for acquiring language. Therefore we take this presupposition to be at least approximately true.

2.4.3 No Noise in the Input

In addition, we will assume all input sentences that the child receives to be grammatically correct: all input does indeed conform to one single grammar for the target language. Again, this is an obviously false assumption. For instance, parents of very young children are notorious for the ungrammatical "goo-goo-gaa-gaa"-like way they talk to their infant. This is not a problem for us, however, since our objective here is to show that unbiased language learning is computationally intractable. Since learning with noise is at least as hard as learning without noise, it will be sufficient for us to show that *noiseless* unbiased learning is already too hard.

---

belong to the language generated by G, we may tentatively assume that the child has acquired the grammar G. §4 of Chapter 1 of [Cho65] has more on this.

2.4.4 PAC Learnability Is the Right Analysis

The final assumption is that polynomial-time PAC learnability (or, somewhat more liberally, polynomial-time PAC predictability) provides an appropriate analysis of learnability. That is, we will take it that a concept class is learnable for a child if and only if the child "has" an efficient (polynomial-time) PAC learning algorithm for that class. For language learning, this means that a class of possible languages is learnable if and only if the child's language acquisition mechanism is a polynomial-time PAC learning algorithm for that class.

Some readers may feel this to be too strong a requirement. After all, PAC learnability is a worst-case analysis over *all* possible probability distributions on the domain. Do we really require that a child be able to learn a language with all possible distributions over the examples (some distributions are pretty weird)? Maybe not. Maybe the child's language acquisition algorithm only works for certain probability distributions, for instance those under which the most often used sentences are also more probable to appear as input. But that would mean that the child's learning algorithm is already biased to particular probabability distributions, and that it would not be able to learn its native language under some other distributions. Either way, whether PAC learnability is the right analysis or not, we have established the main objective of this chapter: a child must have a certain bias which directs the way it acquires language.

Another feature of PAC learnability which may seem too strong, is the use of the confidence parameter δ and the error parameter ε. Can we really set δ and ε to arbitrarily small values, and be sure that a child will, with probability at least $1-\delta$, learn a language (actually, a grammar for that language) which has error less than ε compared to the target language? Again, maybe not. Maybe this is indeed too much to ask. Nevertheless, it seems fair to assume that giving a child more example sentences, as well as more time to think those sentences over, will *increase* the probability that it learns an approximately correct language and will *decrease* the number of errors the child makes. From this I conclude that the requirements of PAC learnability are at least right *in spirit*, even though the technical details of those requirements might be somewhat too strong.

2.5 Formal Analysis of Language Learnability

2.5.1 The PAC Setting for Language Learning

We will now tune the PAC setting of the previous chapter to language learning. Let us take as our domain X all possible sentences, i.e., all finite sequences using symbols from some fixed alphabet Σ. A *language* is then a concept over X, and a concept class (or *language class*) is a set of languages. Note that grammars can be used to *represent* languages, in the sense of Section 1.6.1, as follows. A grammar G *represents* (or is a name of) a language L if L equals the set of sentences generated by G: the grammar G is a name of the language $L(G)$. We will call this representation the *grammatical* representation. Note that since a language can usually be generated by more than one grammar, most languages will have more than one name in this representation.

2.5.2 A Conjecture

In Section 2.4, we have made the assumption that all children have the same algorithm for learning their native language from input sentences. Let us call this algorithm C (for Child). No doubt C is extremely complex, and no one knows exactly what it looks like.[11] However, this need not detain us here—the abstract assumption of the existence of this algorithm is sufficient for our purposes.

The algorithm C is an algorithm for learning languages from example sentences. Since children all over the world are able to learn their native language, C must be an algorithm that can learn at least all existing natural languages. Moreover, it is well known that *most* children, when provided with sufficiently many example sentences of what will become their native language, learn that language *almost perfectly*. Thus, when a child is presented with example sentences from some natural language, it will *probably* learn that language *approximately correctly*. I will take this as strong evidence for the conjecture that *C is a PAC learning algorithm* for the class of all existing natural languages. Furthermore, children learn their language quite fast and without much visible effort, usually within only a few years—children certainly outperform present-day language-processing computers when it comes to language learning. This is particularly fast when compared to the time and effort humans generally need to acquire competence in other complex areas (for instance, learning mathematics, or learning how to play the piano) to the same level of perfection. Thus it appears that children not only learn language probably approximately correctly, but that they do so quite *efficiently* as well. Therefore, we will strengthen our conjecture by assuming that C is an *efficient* (i.e., polynomial-time) PAC learning algorithm for the class of all existing natural languages. Finally, there is no need to assume that the existing natural languages are the *only* languages that our algorithm C can learn efficiently: any language sufficiently similar to the existing natural languages will be learnable by children as well.[12] Accordingly, we will make the following claim:

> There exists a language class $\mathcal{L}$, containing (probably among others) all existing natural languages, such that C is a polynomial-time PAC algorithm for $\mathcal{L}$.

Our aim in the following sections is to find *constraints* on $\mathcal{L}$, using arguments from computational learning theory. Specifically, it will be shown that $\mathcal{L}$ *cannot* be the set of all context-sensitive languages.

2.6 The Learnability of Type 4 Languages

In this and the following sections, we will investigate the learnability of the different levels in the Chomsky hierarchy. Actually, it will turn out that without

[11] Steven Pinker's [Pin84] contains a very elaborate and ambitious proposal as to the actual learning mechanisms used.

[12] Specifying what "*sufficiently* similar to the existing natural languages" means is more or less the same as specifying Universal Grammar.

additional help (e.g., the ability to make membership queries), even the simplest class in the Chomsky hierarchy, the class of Type 3 languages, is not efficiently learnable.

In fact, we can define an even simpler type "on top of" the Chomsky hierarchy, which is still not efficiently learnable. Recall that a language is of Type 3, or *regular*, if it can be generated by a grammar containing only productions of the following forms:

$\mathbf{A} \rightarrow$ a
$\mathbf{A} \rightarrow$ a$\mathbf{B}$

Such a grammar may be circular or recursive, in the sense that, for instance, it contains productions $\mathbf{A} \rightarrow$ a$\mathbf{B}$, $\mathbf{B} \rightarrow$ b$\mathbf{C}$, and $\mathbf{C} \rightarrow$ c$\mathbf{A}$. We can restrict the regular languages by banning such recursion. Formally, this is defined as follows:

Definition 2.1 Let G be a Type 3 (regular) grammar, and N be the set of non-terminals in G. We say there is a *chain* in G from $\mathbf{A} \in N$ to $\mathbf{B} \in N$, if one of the following holds:

1. G contains a production of the form $\mathbf{A} \rightarrow$ a$\mathbf{B}$.
2. There exists a chain from $\mathbf{A}$ to some $\mathbf{C} \in N$, and G contains a production of the form $\mathbf{C} \rightarrow$ a$\mathbf{B}$. ◇

Definition 2.2 A Type 3 grammar G, with set of non-terminals N, is of *Type 4* (or *non-recursive*), if there does not exist a chain in G from any $\mathbf{A} \in N$ to $\mathbf{A}$. A language is of Type 4 if it can be generated by a Type 4 grammar. ◇

As the next theorem shows, the class of Type 4 languages is fairly simple indeed:

Theorem 2.1 *A language L is of Type 4 iff L is finite.*

Proof

$\Rightarrow$: Suppose L is generated by Type 4 grammar G. Note that if there is a derivation of a sentence $s = a_1 \ldots a_k$ from G, then G contains productions $\mathbf{S} \rightarrow$ a$_1\mathbf{A}_1$, $\mathbf{A}_1 \rightarrow$ a$_2\mathbf{A}_2$, $\ldots$, $\mathbf{A}_{k-2} \rightarrow$ a$_{k-1}\mathbf{A}_{k-1}$, $\mathbf{A}_{k-1} \rightarrow$ a$_k$. Then there is a chain from $\mathbf{S}$ to any $\mathbf{A}_i$, and a chain from $\mathbf{A}_i$ to $\mathbf{A}_j$ whenever $i < j$. Now, assume L is infinite. Then there is a sentence $s \in L$, such that s cannot be generated without using some production $\mathbf{A} \rightarrow$ a$\mathbf{B}$ more than once. But then there would be a chain from $\mathbf{A}$ to $\mathbf{A}$, contradicting the assumption that G is of Type 4.

$\Leftarrow$: Suppose $L = \{s_1, \ldots, s_k\}$ is finite. It is easy to see that L can be generated by a Type 4 grammar, using a separate set of non-terminals for each s_i. □

The reader may wonder why we have not used a more general definition of non-recursive languages. After all, a similar definition of 'chain' and 'non-recursive' may be given for grammars of arbitrary type. However, it can in fact easily be shown that if we generalize the definition, then it still holds that

any non-recursive grammar, even one of Type 0, can generate only a finite (and hence Type 4) language. Thus it is no real restriction to define non-recursiveness for Type 3 languages only.

Alternatively, we might have defined Type 4 languages more generally by limiting the number of recursive applications of productions to some number k, instead of banning recursion altogether. That is, we might have defined a *k-recursive* language as a language generated by a Type 3 grammar under the constraint that for any $\mathbf{A} \in N$, a derivation of a sentence uses at most k productions that have $\mathbf{A}$ as left-hand side. (Note that this would not be a restriction on the grammar, but on the way the grammar is used to generate sentences.) But again, this is no real restriction, for it can be easily shown that such a k-recursive language will be finite, and hence already in the class of Type 4 languages as formally defined above.

Since some regular grammars generate infinite languages, for instance $G = \{\mathbf{A} \to \mathrm{a},\ \mathbf{A} \to \mathrm{a}\mathbf{A}\}$, it follows that the class of non-recursive languages is a proper subset of the class of regular languages.

Unfortunately, even the very simple class of all non-recursive languages is not efficiently learnable:

Lemma 2.1 *The class of non-recursive languages is not of polynomial VC-dimension.*

Proof Let $\mathcal{F}$ be the concept class of all non-recursive languages, over some fixed alphabet Σ which contains s characters. Then the number of sentences of length i is s^i, and the number of sentences of length at most n is

$$s^1 + s^2 + \ldots + s^{n-1} + s^n \geq s^n.$$

$\mathcal{F}^{[n]}$ is the set of all finite sets of such sentences, so

$$|\mathcal{F}^{[n]}| \geq 2^{s^n}.$$

This implies that $|\mathcal{F}^{[n]}|$ cannot be upper-bounded by a polynomial in n. Hence, by the remarks following Lemma 1.1 in the last chapter, we have that $\mathcal{F}$ is not of polynomial VC-dimension. □

Thus, using Theorem 1.1:

Theorem 2.2 *The class of non-recursive languages is not polynomial-sample PAC learnable.*

Learning languages seems to be very hard indeed! Even the class of all finite languages is not efficiently learnable—there are simply too many such languages. In the next section, we will see how membership queries may help to solve this problem.

2.7 The Learnability of Type 3 Languages

In this section we will investigate the learnability of the class of regular (Type 3) languages. Firstly, since the class of Type 4 languages is a proper subset of the class of Type 3 languages, the negative result of the previous section carries over immediately to Type 3 languages:

Corollary 2.1 *The class of regular languages is not polynomial-sample PAC learnable.*

Furthermore, in Theorem 7.6 of [KV94], it is shown that the class of languages which can be recognized by a *deterministic finite automaton* (DFA) is not efficiently PAC *predictable* in *any* polynomially evaluable representation (under a common complexity theoretic assumption, for which see Section 6.2 of [KV94]). The details of such DFAs need not detain us here. What is important, is that a language is regular if and only if it can be recognized by such a DFA [HU79, Chapter 2]. Hence it follows that the class of regular languages is not efficiently PAC predictable (if membership queries are not available). Since this result holds for any polynomially evaluable representation, it holds in particular when we use grammars to represent languages.[13]

Theorem 2.3 *The class of regular languages is not polynomial-time PAC predictable in the grammatical representation.*

However, in the stronger setting where both equivalence and membership queries are available, the class of languages representable by DFAs *is* efficiently exactly learnable (Theorem 8.1 of [KV94]). This result is due to Angluin [Ang87b]. Since learning with equivalence queries implies PAC learning (see Section 1.7.3), and a DFA can easily be converted into a regular grammar, we have the following result:

Theorem 2.4 *The class of regular languages is polynomial-time PAC learnable with membership queries in the grammatical representation.*

Let us take a step back to the real world for a moment. After all, we are analyzing the learnability of languages by human beings, in particular by young children. What would membership queries be for a child? A membership query is the question whether some particular sentence is a member of the target language. To the extent that a child can ask questions like "Mummy, is this a good sentence?", we can assume it has access to membership queries (also assuming, of course, that mummy gives correct answers). Moreover, young children often implicitly "test" sentences by saying something to see what happens, and to see how its parents react. Such tests may also be seen as a kind of membership queries. Now it is clear that in the initial phase of language learning, a child cannot ask such questions, since the ability to even *pose* those questions

[13] For context-free grammars, the grammatical representation is polynomially evaluable: the problem whether the language generated by some context-free grammar contains some sentence is solvable in polynomial time [HU79, pp. 139–141].

or pronounce the test-sentences already presupposes at least *some* linguistic competence. On the other hand, it seems fair to assume that in more advanced stages of language learning, the child is able to ask such questions. In sum, we may assume that membership queries are not available to the child early in the learning process, but *are* available as soon as the child has acquired some of its native language. Thus the child's algorithm C *might* be an efficient PAC algorithm for the class of regular languages.

2.8 The Learnability of Type 2 Languages

2.8.1 Of What Type Are Natural Languages?

In the last section, we saw that C *might* be an efficient PAC algorithm for the class of regular languages—there are no *computational* barriers for this, if we allow membership queries. But, however this may be, it should be clear that the regular languages are far too simple: full natural languages are much more complex than that.[14] Where in the Chomsky hierarchy should we look for natural language?

It appears that most linguists would agree that a natural language can be described by a context-sensitive (Type 1) grammar. As Allport [All92, p. 107] writes: "...one is not asserting anything particularly remarkable if one claims that all the structures of natural language can be described by a context-sensitive grammar; grammars with great formal power can describe a vast variety of structures, and so it is unsurprising if natural language structures are all members of such an unrestricted set." Whether or not natural languages can be described by context-*free* (Type 2) grammars seems to be a matter of dispute. On the one hand we have Postal, who claims to prove that the Mohawk language is not context-free, and who provides fairly strong arguments for the claim that the English language is not context-free either [Pos64]. Brandt Corstius disagrees with Postal's proof, but provides his own proof (in Dutch) that Dutch is not context-free [BC74, Stelling 4.9]. His idea applies to English as well. Consider the sentence scheme "These physicists, philosophers,..., from, respectively, the U.S., Holland,..., are, respectively, super-smart, smart,...". The point is that an instance of this scheme is grammatical only if the sequences filled in on the three dotted parts agree in the number of terms. For

[14] Certain rash claims by the supervisor of the present thesis notwithstanding: in [Lok91] it is claimed that human beings are deterministic finite automata, which suggests that human natural languages are regular (Type 3). This claim is based on the obviously true assumption that humans have a limited processing capacity (lifetime and memory), and hence cannot comprehend sentences of more than, say, one billion words. In fact, assuming such a maximal length of sentences renders natural languages finite, and hence only of Type 4! (Moreover, such a length-bound would make the class of possible languages learnable as well, see Section 2.10.)

However, we should distinguish between the sentences human beings can actually comprehend or process (which is part of *performance*), and those that they would consider grammatical (part of *competence*). It might well be that the set of sentences any human being can process is finite, and hence of Type 4. Still, most native English speakers would consider the infinite class of "respectively"-sentences considered below to be acceptable (we can query the whole set in a single question to a native speaker: "Do you consider all such sentences acceptable?"), which would make natural language at least context-sensitive.

instance, "These physicists, philosophers, sociologists, from, respectively, the U.S., Holland, Belgium, are, respectively, super-smart, smart, not too dumb" is grammatical, but "These physicists, philosophers, from, respectively, the U.S., Holland, and Belgium, are, respectively, super-smart, smart, not too dumb, and totally silly" is not. It can be shown that a language containing such a fragment is not context-free (abstractly, the language $\{a^n b^n c^n \mid n \geq 1\}$ is not context-free).

On the other hand, more recently Gazdar et. al. [GKPS85] have conjectured that English *can* in fact be described by so-called generalized phrase structure grammars, which are actually equivalent to context-free languages: a language can be generated by a generalized phrase structure grammar iff it can be generated by a context-free grammar. Thus it is not quite clear if we need context-sensitive (Type 1) grammars for natural language, or whether context-free (Type 2) grammars suffice.

However this may be, it is clear that we have to investigate the learnability of languages more complex than Type 3. In this section we look at Type 2 languages, in the next at Type 1.

2.8.2 A Negative Result

Because the class of Type 2 languages is a superset of the class of Type 3 languages, Theorem 2.3 immediately carries over to Type 2:

Corollary 2.2 *The class of context-free (Type 2) languages is not polynomial-time PAC predictable in the grammatical representation.*

What happens if we allow membership queries? Will this make these classes efficiently learnable? To my knowledge, no answer to this question has appeared in the literature, and neither have I been able to prove it myself. However, my conjecture would be that the class of context-free languages is not polynomial-time PAC predictable, even given an oracle for membership queries.

2.8.3 k-Bounded Context-Free Languages Are Learnable

Angluin [Ang87a] has proved a positive result for so-called *k-bounded* context-free languages. A context-free grammar is *k-bounded* if each of its productions has at most k non-terminals (and any number of terminals) in its right-hand side. A context-free language is k-bounded if it can be generated by a k-bounded context-free grammar. For example, the toy grammar from Section 2.3.2 is 2-bounded.

Angluin's result assumes that there is not only a target *language* L, but also a particular target *grammar* G for that language. The result depends on the presence of an oracle for *non-terminal* membership queries. Such an oracle takes a string x and a non-terminal $\mathbf{A}$ as input, and anwers 'yes' if x can be generated from the productions in G using $\mathbf{A}$ as starting symbol, and 'no' otherwise. Note that an ordinary membership query for the target language is a non-terminal membership query with $\mathbf{A}$=$\mathbf{S}$. For fixed k, the class of k-bounded

context-free languages is polynomial-time identifiable from equivalence and non-terminal membership queries, and hence polynomial-time PAC learnable from non-terminal membership queries alone.

Angluin's result depends on a *fixed* k: in order to function properly, the algorithm that learns k-bounded context-free languages has to know in advance what k is. Actually, since any context-free grammar can be put in Chomsky normal form [HU79, pp. 92–94], where each production has the form $\mathbf{A} \rightarrow$ a or $\mathbf{A} \rightarrow \mathbf{B}\ \mathbf{C}$, any context-free language is 2-bounded. Thus there exists a polynomial-time PAC algorithm for the class of all context-free languages, *if* (and this is a very unrealistic "if") the target grammar is always in Chomsky normal form and the algorithm can make non-terminal membership queries with regard to this grammar.

How realistic are non-terminal membership queries, from the point of view of a young child? In the previous section, we saw that ordinary membership queries correspond to questions like "Mummy, is this a good sentence?" Non-terminal membership queries are similar, except that the child may now pose questions about any grammatical category. That is, it may pose questions like "Mummy, is this a noun?", "Is this a prepositional phrase?", "Is this an auxiliary verb?", etc. The ability to pose sensible questions about nouns, prepositional phrases and what not, presupposes quite sophisticated grammatical knowledge on the part of the child, and the ability to *answer* such questions presupposes even more sophisticated grammatical knowledge on the part of mummy. Therefore, it seems to be rather unrealistic to attribute the ability to make such non-terminal membership queries to young children.

2.8.4 Simple Deterministic Languages are Learnable

Another positive result for a subset of the context-free languages has been established by Ishizaka. It is known that any context-free language can be generated by a context-free grammar in *Greibach normal form*. Here each production has the form $\mathbf{A} \rightarrow$ a $\mathcal{N}$, where 'a' is a terminal and $\mathcal{N}$ is a string of zero or more non-terminals [HU79, Theorem 4.6]. A *simple deterministic grammar* (SDG) G is a grammar in Greibach normal form, such that for any terminal 'a' and any non-terminal $\mathbf{A}$, G contains at most one production of the form $\mathbf{A} \rightarrow$ a $\mathcal{N}$. A *simple deterministic language* (SDL) is a language generated by an SDG. The class of SDLs is a proper subset of the class of context-free languages, and properly includes the class of regular languages.

Ishizaka [Ish90] provides a polynomial-time algorithm which exactly identifies any SDL from membership queries and extended equivalence queries.[15] However, it should be noted that even though the grammar that Ishizaka's algorithm learns does indeed generate the target SDL, that grammar will not always be an SDG.

[15] When we are learning a class of languages $\mathcal{L}$, with an associated class of grammars $\mathcal{G}$ representing those languages, an ordinary equivalence query may only query the correctness of a grammar from $\mathcal{G}$. An *extended* equivalence query may query the correctness of any grammar.

2.9 The Learnability of Type 1 Languages

Here we will look into the learnability of the class of context-sensitive (Type 1) languages, where negative results abound. Firstly, as before, Theorem 2.3 carries over immediately to lower types:

Corollary 2.3 *The class of context-sensitive (Type 1) languages is not polynomial-time PAC predictable in the grammatical representation.*

Furthermore, in this case membership queries will no longer help us. The reason for this is actually quite simple. Efficient learning can only be done in a polynomially evaluable representation, and for context-sensitive languages, the grammatical representation is not polynomially evaluable: in general, the problem of deciding whether a language generated by a context-sensitive grammar contains a particular sentence is $\mathcal{NP}$-complete, and therefore (in all likelihood) not solvable in polynomial time.[16] Thus we have the following result:

Theorem 2.5 *If $\mathcal{P} \neq \mathcal{NP}$, then the class of context-sensitive languages is not polynomial-time PAC predictable with membership queries in the grammatical representation.*

This result may be strengthened somewhat, since even for certain restricted kinds of context-sensitive grammars, the problem of deciding whether a a grammar generates some sentence remains $\mathcal{NP}$-complete. In particular, Aarts proves this for so-called *acyclic* context-sensitive grammars, which lie properly between context-sensitive (Type 1) and context-free (Type 2) grammars. For the details of such acyclic grammars and the proof of the $\mathcal{NP}$-completeness result, see Chapter 4 of [Aar95]. Aarts' result implies that the class of acyclic context-sensitive languages is not polynomial-time PAC predictable with membership queries in the grammatical representation either (assuming $\mathcal{P} \neq \mathcal{NP}$).

2.10 Finite Classes Are Learnable

Finally, let us end our investigations of formal language learnability with a positive result:

[16] Very briefly and informally: $\mathcal{P}$ is the class of all problems solvable in polynomial time (more precisely, solvable in time polynomial in the size of the problem instance), and $\mathcal{NP}$ is the class of all problems for which the correctness of a solution can be verified in polynomial time. A problem Π is $\mathcal{NP}$-complete if Π is a member of the class $\mathcal{NP}$, and if every other problem in $\mathcal{NP}$ can be "translated" to Π in polynomial time. A particular $\mathcal{NP}$-complete problem is polynomially solvable iff *all* $\mathcal{NP}$-complete problems are polynomially solvable. It is known that $\mathcal{P} \subseteq \mathcal{NP}$, and if $\mathcal{P} \neq \mathcal{NP}$, then the $\mathcal{NP}$-complete problems are not solvable in polynomial time. The inequality of $\mathcal{P}$ and $\mathcal{NP}$ has been (and still is) the main open question in complexity theory, but it is conjectured by virtually everyone that the inequality holds. It is in fact a common working assumption that the inequality holds, and, hence that the $\mathcal{NP}$-complete problems are not solvable in polynomial time. It would have momentous consequences (for instance on encryption methods) if this turned out otherwise.

See [GJ79] for an introduction into $\mathcal{NP}$-completeness, and p. 271 of that book for the particular $\mathcal{NP}$-completeness result about context-sensitive grammars mentioned here.

Theorem 2.6 *Any finite class of Type 0 languages is polynomial-time PAC learnable in the grammatical representation.*

Proof Let $\mathcal{F} = \{L_1, \ldots, L_k\}$ be a finite class of Type 0 languages, and let $G_1, \ldots, G_k$ be Type 0 grammars for those languages, respectively. This class is easily seen to be identifiable from at most k equivalence queries: an algorithm that makes an equivalence query for each G_i, ignoring the returned counterexamples, will do the job. As soon as the oracle answers 'yes' on some G_i, we have identified L_i as the target language. Since k is fixed, it follows that $\mathcal{F}$ is polynomial-time identifiable from equivalence queries. This implies that $\mathcal{F}$ is polynomial-time PAC learnable as well. □

In some respects, this is a very interesting result, because if we ignore variation in the lexicon, then Chomsky's current principles-and-parameters theory only allows a *finite* number of distinct natural languages: there are only finitely many parameters, each of which can take on only a finite number of distinct values, so the number of allowed sets of principles is finite (see [Cho86, p. 146] and [Cho91, p. 26]). If we could somehow limit the set of allowed lexicons, it would follow that the class of natural languages is polynomial-time PAC learnable.

Note, however, that the polynomial-time PAC learning algorithm for $\mathcal{F} = \{L_1, \ldots, L_k\}$ must "know" in advance grammars $G_1, \ldots, G_k$ that generate the languages in $\mathcal{F}$. Thus if the child's algorithm C is such an algorithm for a finite class of languages, the main claim of this chapter is vindicated: human language learning needs bias (in this case, pre-knowledge of the k possible languages).

Note also that putting an upper bound on the length of sentences makes the set of possible sentences finite, which in turn makes the set of all possible languages finite, and hence polynomial-time PAC learnable. There may actually be some biological truth in an upper bound on the length of sentences that can be processed, since any sentence-processing human being will have a limited memory and lifetime (see also footnote 14). However, the positive learnability result in case of such a length-limitation is an artefact of our definitions rather than a positive result about learnability in practice. The result follows from the fact that a constant-bounded number of computational steps is polynomially-bounded, and therefore considered efficient by our definitions, no matter how large the constant bound actually is.

For instance, if we use the simple binary alphabet $\{0, 1\}$ and limit the length of a sentence to a 1000 characters, there are $2^{1000} - 1$ sentences, and hence $2^{2^{1000}-1}$ possible languages. This class is learnable according to our definitions, because an algorithm needs at most a constant—and hence obviously polynomially-bounded—number of equivalence queries to identify a language from this set. Unfortunately, this constant ($2^{2^{1000}-1} \approx 10^{10^{300}}$) may be considered infinite for all practical purposes, since it is slightly larger than the number of particles in the universe.

2.11 What Does All This Mean?

Let us now take stock. We have seen a whole bunch of formal theorems about the learnability and non-learnability of various classes of formal languages. What does all this mean for language acquisition by children? Well, it follows from the previous results that whatever the child's learning algorithm C is, it cannot be an efficient learning algorithm for all context-sensitive languages (not even for all *acyclic* ones), let alone for all possible Type 0 languages. Accordingly, the class $\mathcal{L}$ of languages mentioned in the conjecture of Section 2.5.2 *must necessarily be a very restricted class of languages.* Furthermore, learning algorithms for such restricted classes will only work if they "know" in advance what they are looking for, that is, if they know the class of languages that the target language comes from. In other words, our algorithm C needs to know in advance which class of languages are possible target languages—*general-purpose* learning mechanisms, which do not have such pre-knowledge, will not work. This means that children must have a certain linguistic *bias* which enables them to learn certain languages efficiently.

This bias must largely be present "in" the child at the age when language learning commences. Where does it come from? There seem to be only two possible sources for this bias: it can he hard-wired in the brain of the newborn child, and hence innate, or it can be learned in, say, the first year of the child's life, before the process of language learning starts. Probably both sources contribute something to linguistic bias.[17] However, since language does not play a large role in the environment of a very young child, it seems unlikely that the second, non-innate source of bias contributes very much. After all, rattles and mother's milk have very little to do with passivizing transformations and sundry lexical features. From this I conclude that the linguistic bias that children must have is probably for a very large part innate—as Chomsky has argued all along.

Let us call the languages in $\mathcal{L}$ the *Natural Languages* (this includes all existing natural languages), and let us call the set $\mathcal{G}$ of grammars that generate the languages in $\mathcal{L}$ the *Natural Grammars.* Since $\mathcal{L}$ cannot contain all possible Type 0 languages, $\mathcal{G}$ cannot contain all possible grammars. What exactly are the characteristics of the grammars that $\mathcal{G}$ *does* contain? As the quotation from Chomsky on p. 25 indicated, this question is an empirical one, which cannot be fully answered by the mathematical approach of the present thesis. By formal means we can find restrictions on $\mathcal{G}$, such as that it cannot contain all context-sensitive grammars, but these formal tools will not tell us which grammars $\mathcal{G}$ actually *does* contain.

Computational learning theory may provide suggestions as to the class $\mathcal{L}$, but not proofs about its contents. For example, Angluin's positive result for k-bounded context-free grammars might suggest the possibility that $\mathcal{G}$ is the set of all k-bounded context-free grammars, for some fixed k (assuming the child can somehow make non-terminal membership queries). If natural languages

[17] Actually, it has been suggested that language learning already commences *in utero*, which would rule out this second possibility.

are context-free, as some linguists have argued, then this is a significant result. Unfortunately, the formal approach used here will not tell us *which* fixed k is involved here.[18] This k can only be determined from empirical work: write context-free grammars for all existing natural languages, and see by which k they are all bounded. Similarly, the result of the previous section might suggest that $\mathcal{G}$ is some *finite* set, but no merely mathematical work will tell us *which* finite set of grammars $\mathcal{G}$ actually is.

The restriction of our language acquisition procedures (i.e., algorithm C) to languages with some Natural Grammar enables human beings to acquire their native language quite efficiently. However, there is one disadvantage with possibly far-reaching consequences. Namely, learning a language that does *not* conform to Natural Grammar might be exceedingly hard for us humans. This suggests that languages of beings whose "hard-wiring" or "cognitive structure" is quite different from our own (e.g., martians and possibly some animals), would not be learnable for us—and hence we would not be able to understand and communicate with those beings.[19]

2.12 Wider Learning Issues

The previous sections applied computational learning theory to the acquisition of languages by children. To sum up the argument:

1. The class of natural languages must be efficiently learnable, because most children successfully acquire one or more languages.
2. Computational learning theory shows that only very restricted classes of grammars are efficiently learnable. In particular, the classes of all Type 0 or even Type 1 languages are *not* efficiently learnable.
3. Hence the class of natural languages cannot be the class of all languages; it must be some very restricted class.
4. Children must have some pre-knowledge (or bias) of these restrictions upon the class of natural languages in order to be able to learn. This bias is probably largely innate.

However, language learning is just one example where learning from examples takes place. Young children are exceedingly good at learning many other things besides language as well, such as learning to recognize faces, learning how objects usually fall, how people walk, which species of animals are dangerous, and so on. Thus far, not much research has been devoted to the PAC learnability of faces or pictures, or of the behaviour of every-day objects. However, a

[18] Since any context-free language can be generated by a 2-bounded grammar in Chomsky normal form, we might argue that $k = 2$. However, this would make the assumption that children can make non-terminal membership queries even more unrealistic, since grammars in Chomsky normal form look very unnatural. It is not very plausible to assume that parents can answer non-membership queries for a "natural" context-free grammar of their language (involving familiar categories like nouns and verbs), let alone if this grammar is transformed into an artificial Chomsky normal form.

[19] This also sheds new light on Wittgenstein's famous dictum "Wenn ein Löwe sprechen könnte, wir könnten ihn nicht verstehen" [Wit53, II.xi, p. 568]: if the lion's language does not conform to our human Natural Grammar, we are probably not able to learn it.

number of general negative results on the learnability of formulas from propositional logic (boolean functions) have appeared.[20] Since propositional logic is a relatively simple system, we may also expect many negative learnability results for the kinds of learning that children engage in every day. On the other hand, children achieve these learning tasks quite efficiently and effortlessly. Thus we are again led to an explanation in terms of innate structures: apparently children are born with a certain bias or pre-knowledge ('knowledge' here taken in a very broad sense) that helps them to learn to deal with the kinds of objects, animals and humans that they are likely to encounter in the early phases of their lives.

There are in fact many empirical results that point in this direction, see for example Steven Pinker's discussion of the innateness of "intuitive mechanics" and "intuitive biology" [Pin94, pp. 420–426]. In Pinker's words: "We all get away with induction because we are not open-minded logicians but happily blinkered humans, innately constrained to make only certain kinds of guesses—the probably correct kinds—about how the world and its occupants work" [Pin94, pp. 153–154].

2.13 Summary

The main conclusion of this chapter: without a strong linguistic bias, children would not be able to learn a language from examples efficiently. Since it is evident that they *do* learn their native language quite fast and approximately correctly, it follows that children must have such a bias. Because it is not very plausible to assume that a child acquires this bias in the short period before the process of language learning starts, it is probably for a very large part *innate*. This vindicates one of the main tenets of Chomskyan linguistics. Similar arguments apply to many other kinds of learning that people in general—and young children in particular—engage in.

[20] See for instance [KV94, Theorem 6.3].

Chapter 3

Kolmogorov Complexity and Simplicity

3.1 Introduction

There is an interesting paradox about words that do or do not apply to themselves (Grelling's paradox). Let us call a word that applies to itself or that is true of itself *autological*. Examples are 'English', which is itself an English word, and 'old', a word which has been in use for many centuries now. Call a word that does *not* apply to itself *heterological*, such as the clearly non-red word 'red', or the non-German word 'German'. Now the question is: *is the word 'heterological' itself heterological?* If it is, then it isn't; if it isn't, then it is—a puzzling paradox indeed.

A clear example of a heterological word is 'simplicity', which is exceedingly complex and hard to explain [Bun62]. Of course, we can teach this notion to someone (that is, to a human being with biases similar to ours) simply by giving some examples, which will usually suffice in practice, but it is very hard to state explicitly and generally what simplicity amounts to. Since the notion of simplicity is a rather important one in philosophy, we are obliged to devote a lot of effort at making it more perspicious: "What the problem of simplicity needs is a lot of hard work" [Goo72c, p. 282]. Fortunately, most of the really hard work has already been done for us in mathematics and computer science, though most philosophers appear to be unacquainted with this work. The fundamental measure of complexity or simplicity that has been developed is called *Kolmogorov complexity* and is the topic of the present chapter.

The basic question to start with is:

> Can we objectively measure the complexity of an object?

A first stab at an answer might be that the complexity of an object is proportonial to the number of its parts. This would require us to be able to identify and count the parts of an object. However, what we recognize as a part of an object is relative to our interests. Thus a car driver would describe a car as consisting of four tires, a steering-wheel, windows etc., while a physicist would

consider it to be built up from particles like protons, neutrons, and electrons.[1] Moreover, it would be rather pointless to call the physicist's description the "more fundamental" or "better" description in general, since this physical description will be quite useless to the ordinary car driver. Therefore, the level of *descriptions* is a more suitable level of analysis of complexity than the level of objects. Accordingly, we will redirect our basic question towards the complexity of descriptions, or, more generally, of *strings* of characters:[2]

> Can we objectively measure the complexity of a string?

At first sight, the complexity of a string appears to be simply its length, i.e., the number of character-tokens it contains, which is not very interesting from a philosophical point of view. However, from simple examples it can already be seen that strings of equal length may diverge widely in complexity. Consider x_1, which is a string of 10,000 1s, and x_2, which gives the results of 10,000 random coin flips (where the ith character of x_2 is 1 if the ith coin flip comes up 'heads', and the ith character is 0 in case of 'tails'). The string x_1, despite being 10,000 characters in length, is actually fairly simple: the string "10,000 1s" fully describes x_1 using only 9 characters. On the other hand, as each coin flip is independent of the others, the shortest description of x_2 will probably be x_2 itself. Thus the complexity of x_1 is much lower than the complexity of x_2.

The thing is, of course, that a string can be represented in many ways, and very "regular" or "simple" strings can be represented very economically. Thus, as we saw, the short string "10,000 1s" can represent the long string x_1. In this vein, we could identify the complexity of a string with the length of its shortest representation. However, the notion of a 'representation' still requires clarification. What does it mean for one string to represent another? Clearly, infinitely many representation-schemes are possible. We could simply write down a representation explicitly as a two-column table, where the strings in the first column line-by-line represent the strings in the second column. This table, however, will grow to infinite length if we want to be able to represent an infinite number of strings.

Ideally, a string would *itself* give something like a "recipe" to generate the string it represents; this would allow us to dispel with the two-column table. For example, the string "10,000 1s" tells us that putting 10,000 1s in a sequence gives us the string x_1 that it represents. Now, the most basic idea of a recipe that we have, is the notion of an *algorithm*; and every algorithm is a *Turing machine*; and every Turing machine can be encoded as a binary string. This gives us the following explication of what it means for one string to represent another:

> String y represents string x if y is the encoding of a Turing machine that generates x (and then halts).

Now we can identify the Kolmogorov complexity of a string x with its shortest representation:

[1] Compare [Wit53, §47].

[2] Similarly, Bunge [Bun62] directs his attention at what he calls *semiotic* simplicity, not at *ontological* simplicity.

> The *Kolmogorov complexity* of a string x is the length of a shortest y such that y represents x.

We use the phrase "*a* shortest y" rather than "*the* shortest y", since in general there may be several distinct Turing machines of the same length that each produce x.

The main aim of this chapter is to extract from the—often highly technical—literature on Kolmogorov complexity those aspects which are of interest to philosophy. Except for the selection of topics and their presentation, no originality is claimed here. The chapter is organized as follows. We start with a precise definition of Kolmogorov complexity in the next section, and state some of its main properties, notably its objectivity up to a constant, its non-computability, and its relation to information theory. After that, we will discuss various philosophical issues where Kolmogorov complexity is relevant. The main application, as the title of this chapter already indicated, lies in a formalization of the notion of *simplicity*, which is omnipresent in philosophy in general and in the philosophy of science in particular. Secondly, Kolmogorov complexity also allows us to clarify the notion of *randomness*, which will be taken up in Section 3.4. Thirdly and finally, the definition of Kolmogorov complexity allows us to give a proof of Gödel's fundamental incompleteness theorem which does not make us of self-referring sentences.

As the reader will notice, the present chapter contains virtually nothing on learning theory, despite the title of this thesis. However, one of the most important applications of Kolmogorov complexity lies in inductive learning. We will defer this till the next chapter. There Kolmogorov complexity will be used to formalize *Occam's Razor*, which says that simple hypotheses are to be preferred over more complex ones.

3.2 Definition and Properties

In this section we will define Kolmogorov complexity and state some of its main properties. The idea to define the complexity of a string as the length of a shortest Turing machine that produces the string was developed independently and for different purposes in the 1960s by three different persons:

- Ray Solomonoff [Sol64] used it in order to define a *universal* probability distribution, which can be used for prediction.
- Andrei Kolmogorov [Kol65, Kol68] primarily introduced the complexity measure named after him in order to study randomness.
- Gregory Chaitin [Cha66, Cha69] defined Kolmogorov complexity for studying complexity as well as randomness.

Though 'Solomonoff-Kolmogorov-Chaitin complexity' might be somewhat more appropriate, the name 'Kolmogorov complexity' appears to have stuck.

3.2.1 Turing Machines and Computability

In the introduction, we came up with the following definition:

> The *Kolmogorov complexity* of a string x is the length of a shortest y such that y represents x.

Here y represents x if it encodes a Turing machine that generates x.

In the next subsection we will make this definition precise. Here we will first explain in some more detail what a Turing machine is, what it does, and what it can do. Turing machines were introduced by Alan Turing [Tur36]. Informally, a Turing machine consists of a table of instructions, which describe how the machine manipulates the symbols on one or more infinite *tapes* of cells. Each instruction tells what the machine should do, given the particular *state* it is currently in and the contents of the tape cell it is currently scanning. Given a state and tape contents, the instruction tells the machine which symbol to write in its current cell, in which direction to move its tape head, and in which state to go next. All the machine does, is follow these instructions step-by-step, until (if ever) it reaches a point where none of its instructions is applicable, and then it halts.

We will restrict the alphabet of symbols allowed on the tapes to 0, 1, and 'blank'. Since anything that can be stated in some language can be encoded in binary, this is not a real restriction. In particular, we can set up a correspondence between the natural numbers and binary strings, for instance the following:

$$(0, \epsilon), (1, 0), (2, 1), (3, 00), (4, 01), (5, 10), (6, 11), (7, 000), \ldots$$

Here ϵ is the empty string. Note that the binary representation of a number n has approximately length $\log n$ bits (here we use logarithms with base 2). We will be a bit informal about the distinction between numbers and the corresponding binary strings, switching back and forth whenever this is convenient; when we speak of some number x, it will be clear from the context whether we mean that number itself, or the corresponding binary string.

A Turing machine T computes a function f from the natural numbers to the natural numbers, as follows. Suppose we start executing T in some initial state, with an initial tape that contains only one binary string, corresponding to the natural number n. If T's execution terminates with some natural number m (in binary) on its tape, we define $f(n) = m$; otherwise, $f(n)$ is undefined. A function is called *total* if it is defined on each element of its domain, so f is a total function if T halts on all natural numbers.

Definition 3.1 A function f from $\mathbf{N}$ to $\mathbf{N}$ that is computed by some Turing machine T is called *partially recursive* or *computable*. If f is total and T halts on all inputs, then f is called *total recursive* or *recursive*. $\diamond$

By the Church-Turing thesis, any intuitively "mechanically computable" function is partially recursive.

Definition 3.2 A set A of natural numbers is *recursive* or *decidable* if there is a recursive function f, such that $f(n) = 1$ if $n \in A$, and $f(n) = 0$ if $n \notin A$.

A is *recursively enumerable* if there is a partial recursive function f such that $f(n) = 1$ if $n \in A$, and $f(n) = 0$ or $f(n)$ is undefined if $n \notin A$. $\diamond$

The intuition behind the latter notion is that a set A is recursively enumerable if there is a Turing machine which outputs a (possibly infinite) sequence containing all and only members of A. We can also define recursiveness and recursive enumerability for sets of other objects, as long as we can encode these as natural numbers. For example, the set of Turing machines that halt is not recursive: there is no algorithm that takes a binary encoding of an arbitrary Turing machine as input, and determines whether this machine halts after a finite number of steps. This is the well-known undecidability of the *halting problem*, due to Turing.

We can also compute functions that have two or more natural numbers as input, and/or two or more numbers as output, by letting the initial or final tape contain two or more natural numbers, separated in some suitable way. An n-tuple of numbers will be denoted by $\langle x_1, \ldots, x_n \rangle$. Using functions with two natural numbers as output, we can also define functions that range over the set of *rational* numbers $\mathbf{Q}$: a (partial) recursive function f from $\mathbf{N}$ to $\mathbf{N}^2$ can also be seen as a (partial) recursive function g from $\mathbf{N}$ to $\mathbf{Q}$, where $g(n) = p/q$ for $f(n) = \langle p, q \rangle$. Using this, we can also compute—or at least approximate—functions ranging over the set of *real* numbers $\mathbf{R}$:

Definition 3.3 A function f from $\mathbf{N}$ to $\mathbf{R}$ is called *enumerable* if there is a recursive function g from $\mathbf{N}^2$ to $\mathbf{Q}$, such that $g(x,k) \leq g(x,k+1)$ for all x, k, and $\lim_{k\to\infty} g(x,k) = f(x)$ (g approximates f from below). Analogously, the function f is *co-enumerable* if it can be approximated from above. Finally, f is *recursive* if there is a recursive g from $\mathbf{N}^2$ to $\mathbf{Q}$ such that $|f(x) - g(x,k)| < 1/k$ for every x and k. $\diamond$

Let $T_1, T_2, \ldots$ be a list of all Turing machines. Each of these computes a partial recursive function, so there is a corresponding list $f_1, f_2, \ldots$ of (all and only) partial recursive functions. A very fundamental concept is the *universal* Turing machine. This is a Turing machine U that can "simulate" all other Turing machines: for every i, there exists a number (or binary string) t_i such that given inputs t_i and a number n, U computes the same function as T_i on n, i.e., $U(t_i, n) = f_i(n)$ for all n. Such a t_i can be called an *encoded Turing machine* or a *program* for U. It is a fundamental result that such universal Turing machines can actually be constructed (there are in fact infinitely many of them). If T_i is a Turing machine and t_i is its shortest binary encoding (relative to U), then the *length of* T_i (relative to U) will be $l(T_i) = l(t_i)$, the string length of t_i.

3.2.2 Definition

Let us fix some particular universal Turing machine T_u, and let f_u be the function from $\mathbf{N}^2$ to $\mathbf{N}$ that T_u computes. If $f_i(y) = x$, then T_u produces x when

its initial input tape contains program t_i and y, i.e., $f_u(t_i, y) = x$. If y is empty, we simply write $f_u(t_i) = x$. Then we can define the Kolmogorov complexity of a string x as the length of a shortest Turing machine that computes x: $K(x) = \min\{l(t) \mid f_u(t) = x\}$. However, for technical reasons we have to make one important addition, namely that we encode the Turing machines in a *prefix-free* way. That is, if t_1 and t_2 are encodings of Turing machines (programs for T_u) then t_1 is a prefix of t_2 only if $t_1 = t_2$ (x is a prefix of y if $y = xz$ for some z). For instance, it cannot be the case that 110 and 1101 are both encodings of Turing machines, since the former is a prefix of the latter.[3] With this condition in place, we can define:

Definition 3.4 Let T_u be some fixed universal Turing machine whose set of programs is prefix-free, and let f_u be the function from $\mathbf{N}^2$ to $\mathbf{N}$ that T_u computes. The *Kolmogorov complexity* of a binary string x is

$$K(x) = \min\{l(t) \mid f_u(t) = x\}.$$

◇

Thus, as promised, the Kolmogorov complexity of a string x is indeed the length of a shortest Turing machine that computes x (starting from an initially empty tape). The Kolmogorov complexity of a natural number n is the Kolmogorov complexity of the binary representation of n.

It is fairly easy to show that there exists a constant c such that for every x, $K(x) \leq l(x) + c$. Informally, a simple computer program like `print x` suffices to generate x. The length of this program will be the length of `print`, which is a constant independent of x, plus the length of x. Thus the Kolmogorov complexity of a string cannot be much larger than its own length. This is how it should be, since a string is a complete description of itself.

3.2.3 Objectivity up to a Constant

In the introduction to this chapter we claimed Kolmogorov complexity to be objective. But doesn't it depend on the particular universal Turing machine T_u we use? Would not a different choice of T_u lead to different complexities? Indeed it would. But still Kolmogorov complexity can be called objective, due to the following *Invariance Theorem*:

Theorem 3.1 (Invariance) *Let T_u and T_v be universal Turing machines, let $K_u(x)$ denote the Kolmogorov complexity of x relative to T_u, and $K_v(x)$ be the Kolmogorov complexity of x relative to T_v. Then there exists a constant c, depending on u and v but not on x, such that for every x:*

$$K_u(x) \leq K_v(x) + c.$$

[3] There also exists a version of Kolmogorov complexity without this requirement. This $C(x)$, discussed in Chapter 2 of [LV97], has some undesirable properties which make it less interesting than the prefix-free version.

This result is a fairly obvious consequence of the fact that any two universal Turing machines can *simulate* each other. Let T_u and T_v be two universal Turing machines. Since T_u is a universal Turing machine, there exists a T_u-program s which computes the same function as T_v. This s can be called a simulation of T_v on T_u. Suppose T, with encoding t, is a shortest Turing machine that computes x relative to T_v (so $K_v(x) = l(t)$). If we want to compute x relative to T_u, then we can take the simulation-program s, feed t into it, and the program s will execute t for us (and hence generate x) *on* T_u. Thus $K_u(x)$ is at most $K_v(x) = l(t)$ plus a constant c which accounts for the length of the simulation program s and some overhead.

The theorem implies that there is a constant c, such that for every x

$$|K_u(x) - K_v(x)| \leq c.$$

Thus, though Kolmogorov complexity depends on the particular universal Turing machine T_u we choose, for larger x and y the relative influence of the choice of T_u becomes negligible. For instance, if $c = 1,000$ and we are dealing with objects of Kolmogorov complexity more than $1,000,000$, then the relative difference $|K_u(x) - K_v(x)|/K_u(x)$ is less than 0.001. This makes Kolmogorov complexity sufficiently objective.

3.2.4 Non-Computability

In general, we can discern two distinct desirable goals in formal analysis. Firstly, it should provide us with a more clear *insight* in the analyzed topic. We will see in the next section how Kolmogorov complexity allows us to supply clear and precise meanings to notions like simplicity and randomness. The second goal of formal analysis is to provide us with useful, practically applicable *tools*. In order for Kolmogorov complexity to be fully applicable, we should be able to find out what the Kolmogorov complexity of a given string is. Unfortunately, this is beyond us (at least, beyond algorithmic means): Kolmogorov complexity is not computable. For a proof, we refer to Theorem 2.3.2 of [LV97].

Theorem 3.2 (Non-computability) *The function $K(x)$ is not recursive.*

However, we are able to *approximate* $K(x)$. There exists a particular recursive function $g(x, k)$ such that if we successively compute $g(x, 1), g(x, 2), g(x, 3)$, ..., then the sequence of numbers we obtain will converge to $K(x)$ from above. For instance, given x and k, g might simulate the first k steps of the first k Turing machines, and output the length of a shortest of these k machines that produces x (if none of the first k machines produces x, let g be some huge, practically infinite number). It is clear that $g(x, k)$ decreases when k grows. Furthermore, if the ith Turing machine is a shortest Turing machine that produces x, say after j steps, then $g(x, \max\{i, j\}) = K(x)$, so $\lim_{k\to\infty} g(x, k) = K(x)$.

Theorem 3.3 (Approximation) *The function $K(x)$ is co-enumerable.*

This result makes "approximate application" of Kolmogorov complexity at least possible in principle (of course, computability of an approximation does not imply *efficient* or *practical* computability of an approximation).

3.2.5 Relation to Shannon's Information Theory

The complexity $K(x)$ of x may be seen as a measure of the amount of *information* inherent in x, since we need a program of at least $K(x)$ bits to reconstruct x. In this subsection we will see how Kolmogorov complexity relates to the older and more famous definition of information given by Claude Shannon [Sha48, CT91]. (We will not use this in the remainder of the thesis, so the reader may wish to skip this section on a first reading.)

Briefly, in Shannon's framework a *message* is a finite sequence of *words*. We will assume each word is a binary string. Each of the possible words $w_1, \ldots, w_k$ has a definite probability (frequency) of appearing. Let $\mathbf{P}$ be the probability distribution over these words. The *entropy* of $\mathbf{P}$ is defined by

$$H(\mathbf{P}) = -\sum_{i=1}^{k} \mathbf{P}(w_i) \log \mathbf{P}(w_i).$$

For example, consider a simple language of three words: '0001', '0011', and '0111', with $\mathbf{P}$-probabilities 1/2, 1/4, and 1/4, respectively. A message in this language is simply a finite sequence of these words, drawn according to $\mathbf{P}$. The entropy is

$$H(\mathbf{P}) = -(\frac{1}{2}\log\frac{1}{2} + \frac{1}{4}\log\frac{1}{4} + \frac{1}{4}\log\frac{1}{4}) = 1.5 \text{ bits}.$$

$H(\mathbf{P})$ is the information content, in bits, of a message of one word. A message of n words then contains $nH(\mathbf{P})$ bits of information.

We can encode messages in this language by assigning each word w_i a particular *codeword* c_i (another binary string), and by encoding each sequence of words as the corresponding sequence of codewords (this can be done in such a way that a bitstring which is a sequence of codewords can always uniquely be decomposed into those codewords again). If we do this in a smart way, assigning short codewords to high-probability words, we can achieve a much more efficient and economical representation than if we represent words simply by themselves. By a fundamental theorem of Shannon's, $H(\mathbf{P})$ is an (almost) reachable lower bound for the length of binary encodings of messages: we can assign each word a codeword in such a way that if we draw a word according to $\mathbf{P}$, then the expected length of its codeword equals $H(\mathbf{P})$ to within one bit (see [CT91, Theorem 5.4.1] or [LV97, Theorem 1.11.2]). Thus, for an optimal encoding of the language above, the expected codelength of an arbitrary message of n words is approximately $1.5n$ bits, whereas if we use each of the three words ('0001', '0011', '0111') simply as its own codeword, the expected codelength is $4n$.

Now, (sequences of) words are simply binary strings, and hence have a Kolmogorov complexity. If we draw a word according to $\mathbf{P}$, then the expected Kolmogorov complexity of this word is $\sum_{i=1}^{k} \mathbf{P}(w_i)K(w_i)$. Surprisingly, this expected Kolmogorov complexity is asymptotically equal to the entropy [LV97, p. 525], provided $\mathbf{P}$ is recursive:

Theorem 3.4 *If* $\mathbf{P}$ *is a recursive probability distribution over words* $w_1, \ldots, w_k$*, then*

$$\lim_{H(\mathbf{P})\to\infty} \frac{\sum_{i=1}^{k} \mathbf{P}(w_i)K(w_i)}{H(\mathbf{P})} = 1.$$

What does this mean? It means that if $\mathbf{P}$ is sufficiently complex (i.e., $H(\mathbf{P})$ is sufficiently large) and we draw a word w according to $\mathbf{P}$, then on average $K(w)$ will be approximately equal to $H(\mathbf{P})$. Consequently, for a long message $v_1 \ldots v_n$ of n words, we can expect $K(v_1 \ldots v_n)$ to approximately equal the information content $nH(\mathbf{P})$ of the message, and we can use the shortest programs that generate strings as approximately optimal codewords for those strings. Nevertheless, despite the fact that the Kolmogorov complexity of a message converges to its Shannon-information content, we agree with Cover and Thomas [CT91, p. 3] that Kolmogorov complexity is more fundamental than Shannon entropy, because it does not depend on a particular probability distribution $\mathbf{P}$.[4]

3.3 Simplicity

From a philosophical point of view, the most important contribution of the theory of Kolmogorov complexity has been to provide us with a precise definition of the *simplicity* of individual strings. Particularly in the 1950s and 1960s, many unsuccessful attempts were made to measure the complexity of theories, especially when formulated in first-order logic. One of the most striking of these was Popper's proposal to identify degree of simplicity with degree of falsifiability (or strength) [Pop59, p. 140]. However, the following example by Goodman [Goo72b, p. 335] shows that simplicity can neither be identified with strength (as Popper wants) nor with safety:

1. All maples, except perhaps those in Eagleville, are deciduous.
2. All maples are deciduous.
3. All maples whatsoever, and all sassafras trees in Eagleville, are deciduous.

Clearly, the second of these hypotheses is the simplest, and is preferable to the others if consistent with the data. However, 3 is stronger than 2, while 1 is safer (i.e., more likely to be true) than 2. Thus neither the strongest nor the safest hypothesis need be the simplest.[5]

Kolmogorov complexity *does* give us a sound and objective quantitative measure for complexity/simplicity. That is:

> A string x (a description, a theory, etc.) is *simple* to the extent that it has low Kolmogorov complexity.

[4] Some relations between Shannon information, Kolmogorov complexity, and information and entropy in physics are described in Chapter 8 of [LV97].

[5] See [Hes67] for an overview of some other approaches at measuring simplicity, and their problems. More recently, Elliott Sober [Sob75] has attempted to equate simplicity with informativeness relative to given questions (a theory is more informative to the extent that it needs less additional information in order to be able to answer a given question).

Not surprisingly, simplicity shows up as a gradual notion here: things are not simple *per se*, but they can be *more* simple or *less* simple. Because, as we have seen, the Kolmogorov complexity of x is objective (to within a constant independent of x), this definition of simplicity is objective as well. The only subjectivity lies in the choice of the particular universal Turing machine we use, but the influence of this choice becomes negligible for larger x. Actually, many philosophers have claimed that simplicity is too subjective and context-dependent to be objectively definable at all. For instance, Lakatos writes

> "No doubt, simplicity can always be defined for *any* pair of theories T_1 and T_2 in such a way that the simplicity of T_1 is greater than that of T_2." [Lak71, p. 131, note 106]

To be sure, this also holds for Kolmogorov complexity. If we want to give a particular string x a very low Kolmogorov complexity, we can achieve this by tinkering with some ordinary universal Turing machine T_u, in such a way that it outputs x if it is given the string 1 as input. Thus the new machine would have x somehow hardwired in its program. Then relative to this new machine, we will have $K(x) = 1$, so we can tinker in such a way that any *particular* string gets a very low complexity. However, if we choose some reasonable universal Turing machine, where no information about particular strings is hardwired, this problem will not arise, and we can stick to the objectivity of Kolmogorov complexity.

Why should we bother about simplicity? Because it is one of the guiding principles of science: simplicity of a theory is generally regarded as a virtue. Scientists generally follow the maxim that simple or elegant theories are to be favoured, both in their practice and in their own theorizing about science. As Quine writes:

> "Consciously the quest [the "sifting of evidence"] seems to be for the simplest story. [...] Simplicity is not a desideratum on a par with conformity to observation. Observation serves to test hypotheses after adoption; simplicity promps their adoption for testing. Still, decisive observation is commonly long delayed or impossible; and, insofar at least, simplicity is final arbiter.
>
> Whatever simplicity is, it is no casual hobby. As a guide of inference it is implicit in unconscious steps as well as half explicit in deliberate ones. The neurological mechanism of the drive for simplicity is undoubtedly fundamental though unknown, and its survival value is overwhelming." [Qui60, pp. 19–20]

Or in Goodman's words:

> "... simplification is the heart of science. Science consists not of collecting particular truths but of relating, defining, demonstrating, organizing—in short of systematizing. And to systematize is to simplify; [...] Science is the search for the simplest applicable theory." [Goo72d, p. 351]

We will see more examples in the next chapter. Given the importance of simplicity, clarifying what makes a simple theory simple, and what makes a simple theory favourable is an important topic in the philosophy of science.

The idea that selecting simple theories is good, however, already brings us to Occam's Razor, to which we will devote the whole next chapter. In this section, we will ignore the prescriptive aspects of simplicity (that simplicity is good), restricting attention to its conceptual aspects. Most important among these: what are its relations to the notions of *elegance* and *beauty*?

In general, if we do not restrict attention to science, what we would call 'simple', 'elegant', or 'beautiful' can diverge widely. For instance, the word 'elegant' often has the connotation of 'slightly superficial', and hence diverges from 'beautiful'. Furthermore, simple works of art need not be beautiful ("Who's afraid of red, yellow, and blue"); conversely, many of the most beautiful pieces of art are highly complex and dense with connotations. On the other hand, simple works of art *can* be very beautiful—examples that come to mind are Mondriaan's abstract paintings and Satie's early piano music. Moreover, both elegance and beauty are irreducibly subjective, whereas simplicity could to a large extent be made objective, as we have seen.[6]

However, when we do restrict attention to the roles of simplicity, elegance, and beauty *in science*, particularly their roles as properties of scientific *theories*, things start to get interesting. Many a scientist or philosopher has used these notions in the same breath, and it is not at all clear how they can be distinguished. Ordinary usage of these terms does not seem to provide us with clear boundaries. On the one hand this is a nuisance, but, on the other, it also leaves us plenty of room to specify these boundaries for ourselves, supplying our own definitions. I would like to propose the following informal definitions:

- A theory is *simple* to the extent that it can be described easily. This can satisfactorily be formalized in terms of low Kolmogorov complexity.
- A theory is *elegant* to the extent that it is simple (in the above sense) and easy to handle.
- A theory is *beautiful* to the extent that it causes feelings of pleasure, delight, reverence, and wonder.

Because of the vagueness of natural language, any boundary will be somewhat arbitrary. Whether these particular boundaries diverge too far from ordinary usage I leave for the native speakers of English to decide.

Our earlier statement that 'simplicity' is largely objective while 'elegance' and 'beauty' are subjective, is clearly in accordance with these definitions. Simplicity can be defined in terms of Kolmogorov complexity, and hence is sufficiently objective. On the other hand, a simple theory is elegant only if it is easy

[6] Even the complexity of works of art is amenable to analysis in terms of Kolmogorov complexity. Literature or musical scores can easily be transformed into bitstrings, which can be assigned a Kolmogorov complexity. Similarly, by treating it as a matrix of colour dots and assigning each dot a number, a painting can be transformed into a bitstring. Very regular paintings will have low complexity, visually very complex paintings will have high complexity.

to handle, easy to use—and this depends, of course, partly on the person who actually uses it: a complex theory is often easy to handle for those having much experience with it, but difficult for first-year students. Thus elegance is partly subjective. That beauty is also subjective will not surprise us: some people feel pleasure, delight, reverence, and wonder very easily, while others remain numb and uninterested even when faced with the theory of relativity. Despite the subjectivity of the elegance or beauty of scientific theories, both are strongly linked to the objectively definable simplicity. In case of elegance, this is immediately apparent from the definition. But also beauty is strongly correlated with simplicity; we will see some examples of this in the next chapter. For one thing, in order to appreciate the beauty of some theory, we should be able to comprehend it fully—which can only be if the theory is sufficiently simple for us to be comprehensible in the first place. Extremely complex theories may sometimes be great predictors, but they will generally not be considered very beautiful.

The subjectivity of beauty in scientific theories also shows up in the *variance* of scientists' aesthetic canons over time. To end this section, let us briefly look at James McAllister's interesting recent account of the role of aesthetical criteria in science [McA96]. According to this account, empirical criteria for theory choice, such as predictive success and consistency with other theories, are supplemented by aesthetic criteria. McAllister mentions five classes of such aesthetic criteria: symmetry, invocation of a model, visualizability/abstractness, methaphysical allegiance, and—most interesting for us—simplicity.[7] Aesthetic value is projected onto an object (for instance a scientific theory) according to such aesthetic criteria or canons. The content and weighing of these criteria are not constant over time, but are (unconsciously) inductively derived from the properties of recent successful theories: our sense of what is beautiful derives from, and varies with, what is successful. Whenever theories are replaced by more successful ones, our aesthetic canons are to some extent adjusted as well.

In McAllister's model, a *scientific revolution* occurs if the aesthetic criteria start lagging too far behind the empirical criteria, and these two sets of criteria start coming in severe conflict. In this case a progressive faction of scientists, which places more value on empiricial than on aesthetic criteria, supersedes a conservative faction that wants to hold on to older theories they perceive as more beautiful. This is corroborated by the fact that many of the truly revolutionary new ideas and theories (such as Kepler's elliptical orbits, or quantum mechanics) were considered by many rather ugly early after their inception,

[7] An important difference between McAllister's handling of simplicity and our own: while we ascribe simplicity to *representations* of scientific theories (namely bitstrings), McAllister ascribes simplicity to those theories *themselves*, not to particular representations of them [McA96, p. 24–26]. The fact that we looking only at representations avoids the problems associated with McAllister's almost Platonic view of theories as abstract entities.

Furthermore, McAllister explicitly rejects using Kolmogorov complexity as *the* measure of simplicity, since there are alternatives, for instance measuring the number of assumptions or the number of variables in a theory [McA96, pp. 119–120]. Nevertheless, we consider simplicity as measured by Kolmogorov complexity to be more fundamental than other measures, because its foundation (the Turing machine as a model of effective computability) is more fundamental and less *ad hoc* than others.

but are considered much more beautiful and elegant now that we have gotten used to them and have been convinced of their empirical success—our aesthetic canons have been adjusted to them.

3.4 Randomness

Actually, despite our focusing on the use of Kolmogorov complexity as a formalization of simplicity, the initial motivation of its development lay elsewhere: namely in the notion of *randomness* of strings. From the point of view of philosophy of science, it is very interesting to specify randomness. After all, scientists aim at finding structure, regularity, causes, etc., in the world, but we cannot rule out *a priori* that some domains have no regularity whatsoever. Intuitively, if some domain possesses no regularity at all, then descriptions that stem from this domain will be *random* strings. Accordingly, ways to recognize randomness are important.

Before the advent of Kolmogorov complexity, several attempts had been made to define necessary and sufficient conditions for randomness, for instance by Von Mises, Wald, and Church.[8] However, each of those definitions included some strings as random which we intuitively would not consider random, and hence failed.

3.4.1 Finite Random Strings

In this subsection, we will characterize the property of randomness of finite binary strings, in the next we will deal with infinite strings. It should be noted that when dealing with finite strings, it is rather arbitrary to fix a sharp boundary between random and non-random finite strings: randomness is a matter of degree here. Furthermore, strings are not random *per se*, but random with respect to a certain probability distribution. For example, a binary string of length 10,000 consisting of about equally many 0s and 1s, distributed in an irregular way over the string, will be fairly random with respect to a probability distribution induced by a fair coin. However, it would be rather surprising if this string were generated by tossing an unfair coin that comes up 'heads' 70% of all tosses, and the string is not random with respect to the distribution induced by the unfair coin.

The way we will define randomness here, is via *tests* for randomness: a string is random if it passes certain tests. For instance, one test for randomness might say that a string that contains much more 0s than 1s is not random with respect to the probability distribution induced by a fair coin, another test might say that a string which starts with a 1,000 1s is not random with respect to that distribution. Thus we could define a string as random if it passes all conceivable tests for randomness. The theory of tests for randomness described below shows that we can actually formalize this. This framework is due to the Swedish mathematician Martin-Löf [ML66], who co-operated with Kolmogorov.

[8] See [LV97, pp. 49–56] for an overview of the various approaches, and why they failed.

A string is random if it passes all conceivable tests for randomness. But what constitutes a "conceivable test for randomness"? Firstly, in order for us to be able to carry out such a test, we should in any case be able to compute or approximate its outcome. Secondly, a test for the randomness of a string x is a test whether x is a "typical" string for that distribution. For instance, 1010100010110 would be a typical outcome for the fair-coin-distribution, while 1111111111111 (13 times 'heads' in a row) would not. The typical strings form a "reasonable majority" of all strings; a majority on which most probability concentrates. Each test is a way of specifying such a majority and of testing whether a given string x belongs to it or not. A string will be called random if it belongs to every "reasonable majority" specified in this way, i.e., if it passes each test. We now first give the definition of a test, and then explain what is intended by this idea of a majority.

Definition 3.5 Let $\mathbf{P}$ be a recursive probability distribution on $\{0,1\}^*$ (the set of finite binary strings). A total function $\delta : \{0,1\}^* \to \mathbf{N}$ is a $\mathbf{P}$-*test* (or a *Martin-Löf test for randomness with respect to* $\mathbf{P}$) if it satisfies the following two conditions:

1. δ is enumerable.
2. $\sum\{\mathbf{P}(x) \mid \delta(x) \geq m, l(x) = n\} \leq 2^{-m}$, for every m and n.

$\diamond$

What is going on here? The first condition is fairly plain: δ can only be an "implementable" or "effective" test for randomness if we can computably approximate it. The second condition requires some more explanation. The idea here is that the elements of $\{0,1\}^*$ that do not belong to some "reasonable majority" of typical strings, are assigned high values by the test δ. The set $V_{m,n} = \{x \mid \delta(x) \geq m, l(x) = n\}$ singles out all strings of length n that are special in having δ-value at least m; this set forms the complement of the "reasonable majority" of typical strings. The second condition in the definition is to ensure that $V_{m,n}$ is indeed the complement of a reasonable majority, by requiring that $V_{m,n}$ becomes ever more more improbable for larger m (equivalently, the probability concentrates on the complement of $V_{m,n}$, which is to contain the typical strings):

$$\begin{aligned}
&\mathbf{P}(V_{1,n}) = \sum\{\mathbf{P}(x) \mid \delta(x) \geq 1, l(x) = n\} \leq 0.5.\\
&\mathbf{P}(V_{2,n}) = \sum\{\mathbf{P}(x) \mid \delta(x) \geq 2, l(x) = n\} \leq 0.25.\\
&\mathbf{P}(V_{3,n}) = \sum\{\mathbf{P}(x) \mid \delta(x) \geq 3, l(x) = n\} \leq 0.125.\\
&\ldots
\end{aligned}$$

Thus, if we draw an arbitrary x of length n according to $\mathbf{P}$, it is very unlikely that x belongs to $V_{m,n}$ for higher m. If x *does* belong to $V_{m,n}$, we have good reason to believe that it is a non-typical string, and we can consider it non-random accordingly. In statistical terms, each $V_{m,n}$ is a *critical region*. If $x \in V_{m,n}$, then we can reject the hypothesis that x is random with *significance level* $1 - 2^{-m}$. Suppose for instance that we have a string x of length n, and

we want to know whether this string is random with respect to a distribution $\mathbf{P}$, using some particular $\mathbf{P}$-test δ. Suppose $\delta(x) = 10$. The second condition in the above definition tells us that the probability that an arbitrary element of length n is a member of $V_{10,n}$ is at most 2^{-10}, which is less than 0.1%, and hence rather improbable. However, our string x *is* a member of this set, so apparently it is a rather special, non-typical string, which does not belong to the "reasonable majority" tested by δ: we can reject the hypothesis that x is random with 99.9% confidence. Thus a $\mathbf{P}$-test gives us information about the "typicalness" of its argument: x tends to be less typical (with respect to $\mathbf{P}$) if $\delta(x)$ is higher (i.e., $x \in V_m$ for higher m).

Recall that we wanted to define a string as random if it belongs to *all* "reasonable majorities", i.e., if it passes *all* tests for randomness (at a certain confidence level). Now, there are clearly infinitely many different $\mathbf{P}$-tests for any probability distribution $\mathbf{P}$, and it would be rather cumbersome to check whether a string x passes each of those tests before we can assign it the predicate "random with respect to $\mathbf{P}$". Instead, it would be much more interesting to have a *single* $\mathbf{P}$-test which only the random strings pass. It is in fact possible to "combine" all possible $\mathbf{P}$-tests into a single $\mathbf{P}$-test. Such a test is called *universal*:

Definition 3.6 A $\mathbf{P}$-test δ_u is *universal* if, for every $\mathbf{P}$-test δ, there exists a constant $c \geq 0$ such that for every string x, $\delta_u(x) \geq \delta(x) - c$. ◇

A universal $\mathbf{P}$-test δ_u is such that no other $\mathbf{P}$-test can find more than a constant amount of regularity in x more than δ_u. It is a fundamental result that for each recursive probability distribution $\mathbf{P}$ there is a universal $\mathbf{P}$-test.

Theorem 3.5 *Let $\mathbf{P}$ be a recursive probability distribution, and $\delta_1, \delta_2, \delta_3, \ldots$ be an enumeration of all $\mathbf{P}$-tests. Then $\delta_u(x) = \max\{\delta_i(x) - i \mid i \geq 1\}$ is a universal $\mathbf{P}$-test.*

The main element in the proof of this theorem (Theorem 2.4.1 of [LV97]) is to show that the sequence $\delta_1, \delta_2, \delta_3, \ldots$ is indeed recursively enumerable. Given that fact, it is fairly easy to show that δ_u as defined here is itself a $\mathbf{P}$-test and has $\delta_u(x) \geq \delta_i(x) - i$, for every i. Hence δ_u is indeed a universal $\mathbf{P}$-test, and can be used to measure the degree of randomness of finite strings: x is less random if $\delta_u(x)$ is higher.

If we fix some universal $\mathbf{P}$-test and some constant c (for instance $c = 1$), then randomness of finite strings with respect to $\mathbf{P}$ can be defined as follows:

Definition 3.7 Let $\mathbf{P}$ be a recursive probability distribution and x a finite binary string. Fix some universal $\mathbf{P}$-test δ_u and constant c. We call x *$\mathbf{P}$-random* if $\delta_u(x) \leq c$. ◇

3.4.2 Infinite Random Strings

In the case of *finite* binary strings we cannot distinguish sharply between random and non-random strings, but in the case of *infinite* strings we can. In this

subsection we will generalize the previous definitions of tests for randomness to the case of infinite strings. We use $\mathcal{B} = \{0,1\}^\infty$ to denote the set of all *infinite* binary strings. If ω is some infinite binary string, we use $\omega_{1:n}$ to denote the finite string consisting of the first n bits of ω. For instance, if $\omega = 110110110\ldots$, then $\omega_{1:5} = 11011$. Because we cannot computably deal with probability distributions on the set of infinite strings (no Turing machine can completely "swallow" an infinite string as input and compute its probability), we have to use something else. We will use a function μ that assigns a number $\mu(x) \in [0,1]$ to any *finite* binary string x, with the following restrictions:[9]

$$\mu(\epsilon) = 1.$$
$$\mu(x) = \mu(x0) + \mu(x1).$$

Here $\mu(x)$ is interpreted as the probability that a string from $\mathcal{B}$ starts with x. The first restriction simply says that any string must start with the empty string (which is fairly obvious). The second says that the probability that an infinite string starts with x equals the sum of the probabilities that it starts with $x0$ or with $x1$ (which is obvious as well, because x can only be followed by a 0 or a 1). Such a function μ is called a *measure* on $\mathcal{B}$.

For a *recursive* measure on $\mathcal{B}$, Martin-Löf's definition of a test for randomness of infinite strings can be stated as follows:

Definition 3.8 Let μ be a recursive measure on $\mathcal{B} = \{0,1\}^\infty$. A total function $\delta : \mathcal{B} \to \mathbf{N} \cup \{\infty\}$ is a *sequential μ-test* (or a *sequential Martin-Löf test for randomness*) if it satisfies the following two conditions:

1. There exists an enumerable total function $\gamma : \{0,1\}^* \to \mathbf{N}$ such that $\delta(\omega) = \sup_{n \in \mathbf{N}}\{\gamma(\omega_{1:n})\}$.
   (Notational remark: the *supremum* of a set A of numbers, denoted $\sup A$, is the maximum of A if A contains a greatest element, and ∞ otherwise.)

2. $\mu\{\omega \mid \delta(\omega) \geq m\} \leq 2^{-m}$, for every m.

$\diamond$

Such tests are called *sequential*, because we can use the function γ to approximate $\delta(\omega)$ by sequentially approximating $\gamma(\omega_{1:n})$ for $n = 1, 2, 3, \ldots$.

Just as in the case of finite strings, a high value for $\delta(x)$ indicates that x is non-typical and hence non-random. Let $V_m = \{x \mid \delta(x) \geq m\}$. The set V_m contains all strings that are identified as special or non-typical in the sense of being assigned a δ-value of m or more. A string x may be said to *pass* the test δ if it is not special for higher m: $x \notin V_m$ for some m, equivalently $\omega \notin \bigcap_{m=1}^\infty V_m$. An infinite string is random if it passes all tests in this way:

Definition 3.9 Let μ be a recursive measure on $\{0,1\}^\infty$, and $\mathbf{V}$ be the set of all sequential μ-tests. An infinite binary string ω is called *μ-random* if it passes all sequential μ-tests: $\omega \notin \bigcap_{m=1}^\infty V_m$ for every δ. $\diamond$

[9] $x0$ is the string which is the concatenation of x and 0; $x1$ is the concatenation of x and 1.

Analogous to the case of finite strings, for each recursive measure μ on the set of infinite strings there is a universal sequential μ-test (see Theorem 2.5.2 of [LV97]).

Definition 3.10 A sequential μ-test δ_u is *universal* if, for every sequential μ-test δ, there exists a constant $c \geq 0$ such that for every infinite string ω, we have $\delta_u(\omega) \geq \delta(\omega) - c$. $\diamond$

Theorem 3.6 *Let μ be a recursive measure, and $\delta_1, \delta_2, \delta_3 \ldots$ be an enumeration of all sequential μ-tests. Then $\delta_u(x) = \sup\{\delta_i(x) - i \mid i \geq 1\}$ is a universal sequential μ-test.*

It is not very difficult to see that the set of infinite strings ω that are random in the sense of Definition 3.9 are exactly the strings for which $\delta_u(\omega)$ is finite. Thus an alternative equivalent definition of randomness is: ω is μ-random iff $\delta_u(\omega)$ is finite. Note that if δ_u and δ_v are two distinct universal sequential μ-tests, then $|\delta_u(\omega) - \delta_v(\omega)| \leq c$, for some constant c and for all ω. Hence $\delta_u(\omega)$ is finite iff $\delta_v(\omega)$ is finite, which shows that it does not matter which particular test we use: each universal sequential μ-test picks out exactly the same set of random strings.

The above paragraphs defined randomness in terms of *tests* for randomness. How does all this relate to Kolmogorov complexity? Kolmogorov complexity allows us to formalize another intuition about randomness: a string is non-random to the extent that it contains many regularities. Very vaguely and abstractly, a regularity is some kind of repetition, the recurrence of a certain pattern. Now, if a string contains some kind of repetition, then we can make use of this to represent the string more efficiently. For instance, a string of length 10,000 that consists of the string '10', repeated 5,000 times, can be represented by the much shorter string "5,000 times '10' ". In other words, if a string contains regularities, we can use these to *compress* the string. Thus a finite string is non-random to the extent that it can be compressed, and is random to the extent that it *cannot* be compressed.[10] Extending this to the infinite case, an infinite string ω may be said to be random if all of its prefixes are incompressible, or at least compressible at most a fixed number of bits. That is, basically the shortest description of its prefixes are those prefixes themselves, and the shortest description of the whole sequence is that sequence itself. The

[10] In the case of finite strings, the definition of $K(x)$ does not enable us to state a clear relation between Kolmogorov complexity and randomness. However, if we temporarily drop the condition that the set of encodings of Turing machines should be prefix-free, and define $C(x|l(x))$ as the length of a shortest Turing machine that generates x, given $l(x)$ on a special input tape, we *can* state such a relation. Define the *randomness deficiency* of a finite string x to be $f(x) = l(x) - C(x|l(x)) - 1$. This deficiency is higher if x is more compressible (more regular). Theorem 2.4.2 of [LV97] states that $f(x)$ is a universal test with respect to the uniform measure. Hence high compressibility indicates non-randomness, and vice versa. (Incidentally, since pseudo random number generators are usually fairly short programs, the "random" numbers that these programs generate will not really be random at all.)

Dennett [Den91] uses Kolmogorov complexity (without mentioning this name) to argue that strings contain real patterns if those strings are compressible.

following fundamental result ([LV97, Theorem 3.6.1]) shows this intuition to be correct if we consider randomness with respect to the *uniform* measure λ, which assigns $\lambda(x) = 2^{-l(x)}$. This λ is a very natural measure, because it distributes probability in an unbiased, uniform way: all distinct prefixes of length n are equally probable.

Theorem 3.7 *An infinite binary sequence ω is λ-random iff there is a constant c such that $K(\omega_{1:n}) \geq n - c$, for every $n \geq 1$.*

3.4.3 An Interesting Random String

In the previous pages we have characterized randomness for finite and infinite strings. As the number of infinite strings is uncountable, whereas the number of Turing machines is only countably infinite, it is clear that there are infinitely many infinite strings that are random with respect to the uniform measure. It is, however, equally clear that we can never effectively describe one. (How could we?—a random infinite string has no finite effective description.) Nevertheless, we can give non-effective descriptions of random strings.

A very interesting example of such a string is the real number Ω. This is defined as follows, where U is some fixed universal Turing machine:

$$\Omega = \sum_{\{t \mid U(t) \text{ halts}\}} 2^{-l(t)}.$$

It can be shown that $0 < \Omega < 1$; in binary expansion, we can write $\Omega = 0.\omega_1\omega_2\omega_3\ldots$, where each ω_i is a bit.[11] We use $\Omega_{1:n}$ to denote $\omega_1\omega_2\ldots\omega_n$, the string of the first n bits of Ω's binary expansion.

This very abstract number is called the *halting probability*, because it is the probability that when we feed a sequence of fair coin flips into the universal Turing machine U, the machine U halts.[12] This can be seen as follows. Consider a monkey who repeatedly flips a fair coin. If the coin comes up 'heads' we write down a 1, and a 0 otherwise, thus producing a growing binary string. As soon as the binary string x is the encoding of a Turing machine, we let the monkey stop flipping coins. Now consider the infinite sequence of all finite binary strings that encode *halting* Turing machines

$$x_1, x_2, x_3, \ldots$$

For each x_i, the probability that the monkey produces x_i is $2^{-l(x_i)}$. (Since the $x_1, x_2, \ldots$-sequence is prefix-free, these events are mutually exclusive, so we can add the probabilities.) For instance, the probability of producing 101

[11] The first bit in a binary expansion corresponds to $2^{-1} = 0.5$, the second bit to 2^{-2}, the third to 2^{-3}, etc. Thus, for example, the binary number 0.11001 corresponds to the decimal $1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} = 0.5 + 0.25 + 0 + 0 + 0.03125 = 0.78125$.

[12] Note that Ω is defined relative to U; distinct universal Turing machines induce distinct Ωs. Furthermore, here we have one of the technical points where the importance of using prefix-free Turing machines shows: if we considered Turing machines encoded in a non-prefix-free way, the sum $\sum_{\{t \mid U(t) \text{ halts}\}} 2^{-l(t)}$ could go to infinity.

is $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = 2^{-3}$. Adding up the probabilities for these x_i, we get the total probability that U will halt on the string produced by the monkey:

$$\sum_{i=1}^{\infty} 2^{-l(x_i)} = \sum_{\{t \mid U(t) \text{ halts}\}} 2^{-l(t)},$$

which is our number Ω.

It is quite remarkable that we can actually define a number which gives the probability that an arbitrary Turing machine halts. Apart from this, Ω has some more interesting properties. Firstly, since it is undecidable whether a given Turing machine halts, it follows that Ω is not recursive. (It is, however, enumerable by a procedure outlined below.)

The most interesting property of Ω, however, is that knowledge of its first n bits (that is, of $\Omega_{1:n}$) enables us to find out which of the first n Turing machines halt. Suppose we know the number $\Omega_{1:n}$, and we want to find out whether some particular Turing machine T, of length n, halts. Consider a program P which simulates all Turing machines: first it executes the first step of the first Turing machine; then it executes the first two steps of the first two Turing machines; then it executes the first three steps of the first three Turing machines; and so on. Then, if we run this procedure forever, each Turing machine which halts will be found to halt after a finite number of steps. Suppose this program P maintains a variable Ω', initially zero. For each Turing machine, of length l, that it finds to halt, let our program P add 2^{-l} to Ω'. Then during the execution of P, Ω' will approach Ω from below. (This procedure can be used to show that Ω is enumerable.) Let P terminate as soon as $\Omega' \geq \Omega_{1:n}$. What do we know once P has halted? If T has been found to halt during P's execution, then of course we know T halts. Now suppose T has not halted during the execution of P. Is it possible that T would halt later on? If so, then $\Omega' + 2^{-n} \leq \Omega$, since T's contribution of 2^{-n} to the halting probability has not yet been incorporated in Ω'. However, since Ω and $\Omega_{1:n}$ can only differ in the $n+1$-th and later bits, we have $\Omega_{1:n} \leq \Omega < \Omega_{1:n} + 2^{-n}$. Thus, if T does not halt during the execution of P but would halt eventually, then $\Omega' + 2^{-n} \leq \Omega < \Omega_{1:n} + 2^{-n}$, which contradicts $\Omega' \geq \Omega_{1:n}$. Accordingly, if T halts during the execution of P then we know it halts; and if it does *not* halt during the execution of P, then we know it *never* halts.

Knowledge of whether or not certain Turing machines halt is tantamount to knowing the answers to many mathematical questions. Consider for instance Goldbach's famous conjecture, which says that every even number is the sum of two primes.[13] We can easily program a Turing machine T which checks if n is the sum of two prime numbers (which each should be smaller than n), first for $n = 0$, then for $n = 1$, then for $n = 2$, and so on, and which halts if it finds an n which is *not* the sum of two primes. This Turing machine T halts iff Goldbach's conjecture is false. Suppose T has length at most n bits, and somehow we know $\Omega_{1:n}$. Using the procedure outlined above, we can find out

[13] To the knowledge of the author, this problem is still open. Fermat's even more famous last theorem (there are no natural numbers $x, y, z \geq 1$ and $n \geq 3$ such that $x^n + y^n = z^n$) is often cited in this context, but this theorem has finally been proved a few years ago.

whether or not T halts, and hence whether or not Goldbach's conjecture is true. And similarly for all other statements that can be encoded in Turing machines of length at most n. Accordingly, if some mathematician had a secret way to access Ω, and sufficient computing power to extract from this whether a given Turing machine halts, he could answer any major question statable in terms of the halting of Turing machines—and become the world's most celebrated mathematician overnight. Unfortunately, the non-computability of Ω tells us that there is no systematic way to access Ω.

Furthermore, it can be shown that Ω is a *random* infinite string, which will become important in the next section: there is a constant c such that

$$K(\Omega_{1:n}) \geq n - c, \text{ for every } n \geq 1.$$

This can be seen as follows. Given $\Omega_{1:n}$, we can compute the set $\mathcal{T}$ of all Turing machines of length at most n that halt. There clearly exists a finite string x that is not computed by any of the Turing machines in $\mathcal{T}$, and hence for which we must have $K(x) > n$. Now, we can define a Turing machine T in which $\Omega_{1:n}$ is somehow encoded, and which uses those n bits to exhibit one such x. This T requires only $K(\Omega_{1:n})$ bits to encode $\Omega_{1:n}$, plus some constant c to compute an x from $\Omega_{1:n}$. Here c does not depend on n. Since $K(x) > n$ and T generates x, we must have that

$$l(T) = K(\Omega_{1:n}) + c > n, \text{ for every } n \geq 1.$$

Accordingly, by Theorem 3.7 we know Ω is random in the sense of passing all sequential tests for randomness with respect to the uniform measure.

The above proof is related to the *Richard-Berry paradox*, which asks for the following number:

> Let x be the smallest number not definable in less than 20 words.

We have just defined x in less than 20 words, despite the fact that x's own definition rules this out! Analogously, the set of all Turing machines of length at most n bits specifies the set of of all strings "effectively definable" in at most n bits. The Turing machine T mentioned above exhibits a string x that is not definable in n or less bits. If its length $l(T)$ were n bits or less, than we had effectively defined x in n or less bits, which would turn the Richard-Berry paradox into a formal contradiction. Hence $l(T)$ must be greater than n, from which the non-compressibility of the prefixes of Ω follows.

3.5 Gödel's Theorem Without Self-Reference

One further application of Kolmogorov complexity lies in a new proof of Gödel's celebrated first incompleteness theorem [Göd31]. Gödel's theorem shows a fundamental limitation of computers: no single computer can fully capture mathematics, in the sense of being able to prove every true mathematical proposition. The philosophical relevance of this theorem shows up for instance in discussions on the philosophy of mind, in particular on theories which take the mind/brain

to be something like a computer. Do human beings have the same limitations as computers? People like Roger Penrose [Pen89] answer negatively, and conclude that the mind/brain must be something more than a computer. Many others, such as Douglas Hofstadter [Hof79], hold on to the idea that the mind/brain is a computer. For them, the importance of Gödels's theorem lies in *self-reference*, in particular the analogy between the self-referring sentence used in the standard proof of Gödel's theorem, and the self-consciousness (the ability to think about ourselves) that is a fundamental aspect of our main/brain. Hofstadter and others take this notion of self-reference or self-consciousness to be more or less the "essence" of mind, and consider Gödel's theorem important because it tells us something fundamental about self-reference. What is interesting, however, is that Gödel's theorem can be proved *without making use of self-referring sentences at all*, thus undermining the analogy with self-consciousness.

3.5.1 The Standard Proof

Let us first spell out Gödel's theorem and its "standard proof" in some more detail. A full introduction to first-order logic lies beyond the scope of the present thesis; we will only explain the things we need, referring to [BJ89] for the many technical details swept under the rug here.

Consider the first-order language of *arithmetic*, the alphabet of which consists of a constant a, a successor function symbol s, two binary function symbols $+$ and $\cdot$, the binary predicate symbol $=$, the usual quantifiers $\forall$ (for all) and $\exists$ (there exists), and connectives $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (if... then), and $\leftrightarrow$ (if and only if), and some variables and interpunction symbols. We assume the reader to be familiar with the usual syntactical rules that specify how terms and formulas are formed from this alphabet. We define a *sentence* as a formula in which all variables are quantified.

Now consider the interpretation of this language which has the set of natural numbers as domain, which assigns the number 0 to the constant a, the successor function $(+1)$ to s, the addition function to $+$ and the multiplication function to $\cdot$, and the equality relation over the domain to $=$. In this interpretation, every term in the language denotes a natural number: a denotes the number 0, $s(a)$ denotes 1, $s(s(a))$ denotes 2, $+(s(s(a)), s(s(a)))$ denotes $2 + 2 = 4$, etc. Furthermore, every sentence in the language has a *truth value* in this interpretation. For instance, $\forall x\ \exists y\ y = s(x)$ is true, because every number has a successor, while $+(s(a), s(a)) = s(s(s(a)))$ is false, because $1 + 1$ does not equal 3. This interpretation of the language of arithmetic is called the *Standard Model* of arithmetic; in the sequel, when we speak of a "true" sentence, we mean a sentence that is true in this Standard Model.

Let us denote the set of true sentences by $\mathcal{T}$. For every sentence ϕ in the language of arithmetic, either ϕ or $\neg\phi$, but not both, is a member of $\mathcal{T}$. Thus $\mathcal{T}$ contains all and only true arithmetical statements. Since large parts of mathematics can be translated into arithmetic, the contents of $\mathcal{T}$ are clearly of the utmost importance. In fact, if we had efficient mechanical access to $\mathcal{T}$, many mathematicians would be out of work. How can we get a grip on $\mathcal{T}$? We can, of course, do what arithmeticians have done for ages: simply try to prove

or disprove some interesting statements with hitherto unknown truth values. From a systematic point of view, however, it would be much more satisfactory to have an adequate *axiomatization* of $\mathcal{T}$. An axiomatization is a recursively enumerable set of sentences. We will call an axiomatization $\mathcal{A}$ *sound* if it implies *only* true sentences (and hence no false ones), and we call $\mathcal{A}$ *complete* if it implies *all* true sentences. Clearly, what we are looking for is an axiomatization which is both sound and complete, i.e., which implies exactly the sentences in $\mathcal{T}$, nothing more and nothing less. In case such an axiomatization exists, we would be able to *enumerate* $\mathcal{T}$: then there is an algorithm which produces (ever longer finite prefixes of) an infinite list containing all and only formulas from $\mathcal{T}$. If we are interested in the truth value of a sentence ϕ, we can simply enumerate this list until we encounter ϕ (in which case ϕ is true) or $\neg\phi$ (then ϕ is false). Thus if we have a sound and complete axiomatization of $\mathcal{T}$, we effectively hold every arithmetical truth in our hands!

Unfortunately, Gödel's theorem tells us that such an axiomatization does *not* exist. The proof that Gödel himself gave, which is repeated in some form or other in most logic books, makes use of a self-referring sentence, similar to the liar's paradox (i.e., the mindboggling sentence "This sentence is false"). Ignoring many technicalities, for which see [BJ89, Chapter 15], this proof runs as follows.

Suppose $\mathcal{A}$ is an axiomatization. We will show that $\mathcal{A}$ cannot be sound and complete at the same time. We can assign to each sentence in the language its own unique natural number, the *Gödel number* of that sentence. Similarly, we can assign Gödel numbers to *proofs* (formal derivations from the axioms of $\mathcal{A}$).[14] Now suppose $\mathcal{A}$ is complete. Then we must be able to express each recursive function. In particular, we can construct a formula $P(x, y)$, with two variables x and y, which is true just in case x is the Gödel number of a proof of the sentence of which y is the Gödel number. Informally, $P(x, y)$ means "x is a proof (from $\mathcal{A}$) of y". Thus for a particular sentence S with Gödel number s, S is provable iff $\exists x\ P(x, s)$ is true.

Now comes the really interesting and technical part: the diagonal lemma, which we will not prove here [BJ89, Lemma 2, p. 173]. It says that for any formula $G(y)$, with variable y, there is a sentence L, with Gödel number l, such that $\mathcal{A}$ implies $L \leftrightarrow G(l)$. Suppose we substitute "$\neg\exists x P(x, y)$" ("y is not provable") for $G(y)$. Then we get that there is a sentence L, with Gödel number l, such that $\mathcal{A}$ implies $L \leftrightarrow \neg\exists x\ P(x, l)$. Thus $\mathcal{A}$ implies that L is true iff L is *not* provable from $\mathcal{A}$! Informally, L can be seen as saying "I am not provable from $\mathcal{A}$". In order for $\mathcal{A}$ to be sound as well as complete, the true statements should coincide exactly with the provable ones. But if L is true then L is not provable, and if L is provable then L is not true! Hence $\mathcal{A}$ cannot be both sound and complete at the same time.

[14] We assume some complete proof procedure is used, so the set of sentences provable from $\mathcal{A}$ is exactly the set of sentences that are logically implied by $\mathcal{A}$. That such complete proof procedures exist is Gödel's completeness theorem from 1930.

3.5.2 A Proof Without Self-Reference

In this subsection we will use Kolmogorov complexity to outline an alternative proof of Gödel's theorem, due to Chaitin [Cha74, Cha75, Cha87], which does not require the construction of a self-referring sentence. Apparently, though self-reference can be used to establish Gödel's theorem, it is not *necessary* for its proof. This sheds new light on the many cases where Gödel's theorem is invoked in discussions about self-reference and self-consciousness.

So, how can we prove Gödel's theorem without self-reference? By making use of the *incompressibility* of the number Ω we saw earlier. We can define a formula $O(n, y)$, with two variables n and y, which is true iff $y = \Omega_{1:n}$, i.e., y is the first n bits of Ω (again, we leave out the technical details of the definition). Thus exactly the following instances of $O(n, y)$ are true:

$O(1, \Omega_{1:1})$
$O(2, \Omega_{1:2})$
$O(3, \Omega_{1:3})$
...

(In order to improve readability, we have written 1 instead of the more correct $s(a)$ in the above formulas, 2 instead of $s(s(a))$, etc.) Now, we can show that for every sound axiomatization $\mathcal{A}$, there exists a number k such that $\mathcal{A}$ cannot imply any true sentence of the form $O(n, y)$ for $n > k$. In other words, the prefixes of Ω longer than k bits are "beyond the grasp of $\mathcal{A}$". From this Gödel's theorem immediately follows, since *each* of the formulas

$O(k + 1, \Omega_{1:k+1})$
$O(k + 2, \Omega_{1:k+2})$
$O(k + 3, \Omega_{1:k+3})$
...

will be *true* but *unprovable*!

Given a sound $\mathcal{A}$, how can we prove the existence of such a k? Suppose such a k does *not* exist. Then for every k, there is an $n > k$ such that $\mathcal{A}$ implies $O(n, \Omega_{1:n})$. There exists a Turing machine which enumerates all logical consequences of $\mathcal{A}$, including $O(n, \Omega_{1:n})$. Let T_n be the shortest such Turing machine, and b_n be its length. Then we can construct from T_n a Turing machine T'_n that generates $\Omega_{1:n}$: this T'_n simply enumerates all logical consequences of $\mathcal{A}$, using T_n, until it finds some $O(n, y)$, and then outputs y, which must be $\Omega_{1:n}$. The length of T'_n will be at most $b_n + 2 \log n + d$, where d is some constant independent of n (the $2 \log n$ term is due to the fact that T_n must know n in order to know what it is looking for). Since T'_n generates $\Omega_{1:n}$, we must have

$$K(\Omega_{1:n}) \leq b_n + 2 \log n + d. \tag{3.1}$$

However, recall from Section 3.4.3 that because Ω is random, there is a c such that

$$K(\Omega_{1:n}) \geq n - c, \text{ for every } n \geq 1. \tag{3.2}$$

Now choose a sufficiently large m such that $b_m + 2 \log m + d < m - c$ and $\mathcal{A}$ implies $O(m, \Omega_{1:m})$. We must have $K(\Omega_{1:m}) \leq b_m + 2 \log m + d$ (because of 3.1)

and $K(\Omega_{1:m}) \geq m - c$ (because of 3.2), which is a contradiction. Hence there must be a k such that $\mathcal{A}$ does not imply any of the true formulas:

$$O(k+1, \Omega_{1:k+1})$$
$$O(k+2, \Omega_{1:k+2})$$
$$O(k+3, \Omega_{1:k+3})$$
$$\ldots$$

Note that while the standard proof provides us with only one example of a true but unprovable formula, namely $L =$"I am unprovable from $\mathcal{A}$", the above proof gives infinitely many such formulas.

In a nutshell: any axiomatization of arithmetic will contain only a finite amount of information, and hence cannot capture all truths about the infinite, incompressible string Ω.

3.5.3 Randomness in Mathematics?

Because Ω is random, it contains no regularities whatsoever (at least, no algorithmically detectable regularities): each of its infinitely many bits is in a way independent of the other bits. This means that the only way to know these bits, is to know them explicitly—no formal system is able to unveil each of them for us. From this, as we saw, Gödel's theorem immediately follows. Our inability to fully capture Ω in a formal system, no matter how complicated, shows a clear limit of formal mathematical reasoning.

Now Ω is a rather outlandish real number, and one may wonder whether mathematicians in general should be much bothered by its existence. However, the number can be translated to the most elementary part of mathematics: elementary number theory. Chaitin has constructed an exponential Diophantine equation E_n (an equation built up from nonnegative integer variables and constants, and a finite number of additions, multiplications, and exponentations) with one parameter n, such that E_n has finitely many solutions if the nth bit of Ω is 0, and E_n has infinitely many solutions if this bit is 1 (see [LV97, pp. 224–225]). Since Ω is incompressible, each of its bits is independent of the others, and hence the existence of finitely or infinitely many solutions of E_n for some particular n is as it were a "brute fact". This is a *very* strong form of Gödel's theorem: formal systems are not even powerful enough to fully capture this single parameterized equation. Chaitin draws rather strong conclusions from this fact:

> "... we see that proving whether particular exponential Diophantine equations have finitely or infinitely many solutions, is absolutely intractable. Such questions escape the power of mathematical reasoning. This is a region in which mathematical truth has no discernible structure or pattern and appears to be completely random. These questions are beyond the power of human reasoning. Mathematics cannot deal with them. Nonlinear dynamics and quantum mechanics have shown that there is randomness in nature. I believe that we have demonstrated in this book that randomness is already present

> in pure mathematics, in fact, even in rather elementary branches of number theory." [Cha87, p. 160].

Here Chaitin's rhetorical step from "no single formal system can deal with all solutions to this equation" to "mathematics cannot deal with them" seems a rather hasty one, since no one has a precise definition of what 'mathematics' is. It certainly seems that mathematics (in the vague sense of "the community of mathematicians and their work") cannot be equated with some single formal system. Moreover, we can agree with Van Lambalgen that the analogy between Ω's randomness and *physical* randomness is "rather farfetched" [Lam89, p. 1398].

Still, the intractability of even a single equation clashes with the usual way we look at mathematics, as a supremely reasonable and reasoned science, where the true facts are true for a reason. Chaitin even goes so far as to suggest that the existence of such randomness should change the way mathematicians work. Rather than formulating axiomatic systems and trying to prove conjectures within these systems, Chaitin argues, mathematicians should work much more in the way of physics. If some conjecture seems plausible but you are not able to prove it (and, by Gödel's theorem, it may actually be true and unprovable at the same time), you may tentatively accept it as a working hypothesis, as a new axiom, experiment with it and see what happens. Indeed, this pragmatic stance—despite being in strong contradiction with the Olympian image many people hold of mathematics—seems to be forced upon parts of mathematics where certain central conjectures turn out to be extremely hard to prove or disprove. An example is the adoption of $\mathcal{P} \neq \mathcal{NP}$ as a working hypothesis in complexity theory (see footnote 16 on p. 39).

3.6 Summary

The Kolmogorov complexity $K(x)$ of a string x is the length of its *shortest effective description*, that is, the length of a shortest Turing machine that generates x, relative to some fixed universal Turing machine U. $K(x)$ is independent (up to a fixed constant) of the choice of U, it is non-computable, and converges to Shannon's measure of information content. It allows us to give an objective formalization of the notion of *simplicity*: a string (theory, description) is simple if it has low Kolmogorov complexity. Furthermore, the *random* infinite binary strings can be identified exactly as those strings whose prefixes are incompressible. Finally, Kolmogorov complexity enables us to show that Gödel's theorem can be proved without self-referring sentences.

Chapter 4

Occam's Razor

4.1 Introduction

Occam's Razor is one of the most important principles of method in science. In its best known formulation, it says that "entities are not to be multiplied beyond necessity". This means that if we can explain some phenomenon in two ways, the first postulating less entities than the second, then we should choose the first. Somewhat more liberally, without focusing on "entities", we may say that Occam's Razor tells us that among the theories, hypotheses, or explanations that are consistent with the facts, we are to prefer simpler over more complex ones. In other words, we should weed out all unnecessary complexity. (Other names for this are the 'principle of simplicity' or the 'principle of parsimony'). How exactly are we to interpret this principle? We can discern at least three different interpretations of the razor:

1. **Methodological.** In this interpretation, selecting simple theories is simply a part of the scientific method, perhaps because simpler theories are easier to work with.

2. **Ontological** (or **metaphysical**). In this interpretation, Occam's Razor is a statement about the world, akin to the laws of physics: it says that the world itself is relatively simple and well-organized, and preferring simple theories is advisable *precisely because the world itself is simple.*

3. **Aesthetical.** In this third interpretation, *beauty* is taken to be a positive indicator of truth: simple theories tend to be beautiful and beautiful theories tend to be true, so selecting simple theories is a good thing.[1]

Note that the second and third of these are stronger than the first: if the world itself is simple (ontological interpretation) or if simplicity indicates truth (aesthetical interpretation), then it is clearly good method to favour simple

[1]

> "Beauty is truth, truth beauty,"—that is all
> Ye know on earth, and all ye need to know.

John Keats, *Ode on a Grecian Urn.*

theories (methodological interpretation). The converse need not hold: the world may be complex and the right theories may be rather ugly, while favouring simple theories may still be good method, because simple theories are easiest for us to deal with.

Occam's Razor plays a prominent role in science, as well as in philosophy of science and philosophy in general; the next section will illustrate this with a number of examples. However, most of those who do the shaving seem to accept the razor more or less as an article of faith, as something intuitively acceptable, which need not or cannot be proved itself. This is a somewhat odd stance. After all, the simplest theory consistent with the available data is not guaranteed to be the right one, and indeed, in many cases it will be wrong. For example, there once were times when circular planetary orbits were consistent with the data available at that time. Would it not be simpler if the planets moved in circles rather than approximate ellipses? Still they don't. Or, as a second example, the overly simple early model of the atomic nucleus: "... one of the most striking examples of how physicists can temporarily be lead astray by the selection of complexes from nature on grounds of simplicity. The case in point is the model of the nucleus built of protons and electrons." [Pai82, p. 326]. So following Occam's Razor in these two cases would lead to false results.

Despite the fact that the razor may yield incorrect results in particular cases, it still remains a quite successful guiding principle in science and elsewhere. Accordingly, an important question is: *what are the justifications for Occam's Razor?* Of course, we can give a fairly shallow inductive argument to the effect that the razor—in each of its three interpretations—has worked quite well in the past, and hence will probably keep doing so in the future. This, however, is rather circular: since considerations of simplicity (i.e., Occam's Razor itself) are crucial for induction, we cannot simply use an *inductive* argument to justify Occam's Razor.

Quine is one of the few who have looked a little closer at why we prefer simple theories. His conclusions:

> "We have noticed four causes for supposing that the simpler hypothesis stands the better chance of confirmation. There is wishful thinking. There is a perceptual bias that slants the data in favor of simple patterns. There is a bias in the experimental criteria of concepts, whereby the simpler of two hypotheses is sometimes opened to confirmation while its alternative is left inaccessible. And finally there is a preferential system of scorekeeping, which tolerates wider deviations the simpler the hypothesis [for instance, changing the simple value 5.2 to 5.23 is more likely to be seen as a refinement (rather than a refutation) of an hypothesis than a change from 5.21 to 5.23, RdW]. These last two of the four causes operate far more widely, I suspect, than appears on the surface. Do they operate widely enough to account in full for the crucial role that simplicity plays in scientific method?" [Qui76, p. 258]

Quine leaves the last question as a real question. But his conclusions are not very encouraging as they stand, since they consider the success of simplicity

(Occam's Razor) to be mainly an artifact of the way we perceive and the way we do science.

However, we can actually do much better than that: in certain formal settings we can, more or less, *prove* that certain versions of Occam's Razor work. This is the topic of the present chapter, which is organized as follows. In the next section, we start with a brief historical overview of Occam's Razor and some examples of its application in science and philosophy. Then in the three ensuing sections, we provide three different formal justifications of Occam's principle. In each of these, simplicity is measured by means of Kolmogorov complexity. Finally, in Section 4.6 we briefly look at what the theory of Kolmogorov complexity has to say on the very possibility of science.

4.2 Occam's Razor

In this section we will first make some historical remarks on the razor, and then describe some examples of its influence in science and philosophy.

4.2.1 History

An appropriate place to start the history of Occam's Razor is, of course, William of Occam himself (also sometimes spelled 'Ockham'). Occam was an English theologian who lived from c. 1285 to c. 1349. Despite his rather exciting life, involving conflicts with the Pope and others, and his relatively modern philosophy, Occam is nowadays mainly remembered for his razor. This is often cited as "Entia non sunt multiplicanda sine necessitate" (entities should not be multiplied without necessity; don't postulate more things than you really need). Notoriously, however, this formulation has not been found anywhere in Occam's actual writings, as was noted by Thornburn [Tho18].[2] Some similar formulations which *can* be found in Occam's work are the following (these and others are cited on pp. 115–117 of [Der93]):

> Pluralitas non est ponenda sine necessitate. (A plurality should not be postulated without necessity.)
> Nulla pluralitas est ponenda nisi per rationem vel experiantiam vel auctoritatem illius, qui non potest falli nec errare, potest convivi. (A plurality should only be postulated if there is some good reason, experience, or unfallible authority for it.)
> Frustra fit per plura, quod potest fieri per pauciora. (It is vain to do with more what can be done with less.)

However, formulations like these were not originally introduced by Occam himself; similar ones can be found in the writings of Occam's teacher, Duns Scotus, as well as in many other medieval philosophers/theologicians. Moreover, as Wil Derkse shows, Occam's Razor might as well be called *Aristotle's Razor*,

[2] As far as we know, the term 'razor' was not used by Occam himself, either. The first known to use the word (the French 'rasoir') in this connection was Pierre Bayle, in the index of his *Pensées* from 1704 (see pp. 97–98 of [Der93]).

because each of its various aspects—the methodological, the ontological, and the aesthetical—can already be found in texts of Aristotle.[3]

To what extent is it still appropriate to call Occam's Razor *Occam's*? From the above quotations from his work, it is fairly obvious that Occam accepted the razor in its *methodological* interpretation. On the other hand, the aesthetical aspect seems to be missing [Der93, p. 135]. It remains to examine to what extent Occam acccepted the *ontological* interpretation of the razor. Admittedly, Occam is well-known for his parsimonious, nominalist ontology. In particular, he denied separate existence to various kinds of abstractions, such as species, relations, causation, motion etc. This is usually taken to be the result of his razor. However, one might question whether the sparseness of Occam's ontology is due to an application of the razor, or to other reasons. In fact, Roger Ariew argues for the latter view. For Occam, the basic principle is not parsimony but *absolute divine omnipotence*. Now suppose, for instance, that relations could have an existence separate from the relata. Since God is omnipotent, this would imply that He could create a relation without creating the relata. But that is absurd, so relations cannot be separate "things" [Ari76, p. 11]. In general, for Occam God's omnipotence bars the ontological assumption that the world itself is simple, since God can—and sometimes *does*—make it more complex than strictly necessary:

> "God does many things by means of more which He could have done by means of fewer simply because He wishes it. No other cause must be sought for and from the very fact that God wishes, He wishes in a suitable way, and not vainly."
> (Occam, as cited on p. 19 of [Ari76]).

Thus for Occam the razor is a methodological, but not an ontological principle: "The razor must be viewed as a restriction on men, not on God or any of its works" [Ari76, p. 24]. Weinberg reaches the same conclusion: "For Ockham there is a principle of Parsimony which applies to human thought, not to the universe" [Wei64, p. 239].

4.2.2 Applications in Science

Here we will illustrate the influence of Occam's Razor in science with some examples.

The starting point of modern science is usually taken to be the work of Copernicus, Kepler, and Galileo, which put cosmology on its present heliocentric feet. Each of these men had a strong preference for simplicity, and can be regarded as adherent of the ontological interpretation of Occam's Razor: nature *itself* is simple and economical. For instance, Copernicus writes:

> "Attacking a problem obviously difficult and almost inexplicable, at length I hit upon a solution whereby this could be reached by fewer and much more convenient constructions than had been handed

[3] See [Boa59] and [Der93, Chapter II] for numerous illuminating citations from Aristotle's work.

> down of old, if certain assumptions, which are called axioms, be granted me." [i.e., Copernicus claims his heliocentic model requires far fewer epicycles than Ptolemy's geocentric one]
> (Copernicus, as cited on p. 50 of [Bur80]).

Many historians and philosophers of science repeat the claim that Copernicus' system is much simpler than Ptolemy's (for instance [Bur80, p. 38]), stating that Ptolemy requires some 80 epicycles, while Copernicus needs only 34.[4] However, much like the usual wording of Occam's Razor, this is a myth which has to be taken "*cum grano salis*, in fact, with the whole saltcellar" [Coh85, p. 111]. Copernicus' epicycles are not quite the same as Ptolemy's, and there are many different ways of counting and comparing the number of cycles each requires. On quite a few of those counts, Copernicus' system comes out more complex than Ptolemy's [Coh85, p. 119].[5]

The real simplification actually only occurred somewhat later, when Kepler dropped Copernicus' assumption that planetary orbits are basically circular (with some additional epicycles thrown in to get empirical adequacy), in favour of ellipsoid orbits. Like Copernicus, Kepler explicitly endorsed simplicity:

> "Natura simplicitatem amat." (Nature loves simplicity.)
> "Natura semper quod potest per faciliora, non agit per ambages difficiles." (Nature does not use difficult roundabout ways to do what can be done with simpler methods.)
> (Kepler, as cited on p. 57 of [Bur80]).

And finally Galileo:

> "Nature... doth not that by many things, which may be done by few." (Galileo, *Dialogues Concerning the Two Great Systems of the World*, Salusbury translation, London, 1661, p. 99, as cited on pp. 74–75 of [Bur80].)
>
> "When, therefore, I observe a stone initially at rest falling from a considerable height and gradually acquiring new increments of speed, why should I not believe that such increases come about in the simplest, the most plausible way? On close scrutiny we shall find that no increase is simpler than that which occurs in always equal amounts." [i.e., gravitation causes constant acceleration]
> (Galileo, as cited on p. 138 of [Der93].)

Isaac Newton hugely contributed to the simplification of physics by bringing very different phenomena (falling bodies, the tides, the movements of the planets, etc.) under the same relatively simple system of general laws, inventing the required mathematics along the way. He, like his predecessors, was explicitly guided by a preference for simplicity. In the third edition of his *Philosophiae*

[4] Copernicus himself claimed greater simplicity for his system in his early *Commentariolus* (1510–1514), but no longer in his later and more famous *De revolutionibus orbium coelestium* (1543).

[5] And his system wasn't more empirically accurate than Ptolemy's, either [Coh85, pp. 116–118]!

Naturalis Principia Mathematica, he included the following as the first of the "Regulae Philosophandi":

> "We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances. To this purpose, the philosophers say, that nature does nothing in vain, and more is in vain when less will serve; for nature is pleased with simplicity, and affects not the pomp of superfluous causes."
> (Newton, *Principles*, II, p. 314, as cited on p. 218 of [Bur80].)

This rule of Newton's is just another formulation of Occam's Razor.

Apart from Newton, Albert Einstein is probably the most important physicist of all times. Not only is he arguably this century's greatest scientist (and certainly the most idolized one), he also was one of those who were most influenced by simplicity. As Derkse writes: "Albert Einstein can be considered as a paradigmatical example of a great scientist who valued simplicity heuristically, sought for simplicity methodologically and believed in simplicity ontologically" [Der93, p. 143]. This is a recurring theme in Abraham Pais' superb biography of Einstein. In fact, Pais concludes that simplicity was the most important driving force behind Einstein's development of relativity:

> "Einstein was driven to the special theory of relativity mostly by aesthetic arguments, that is, arguments of simplicity. This same magnificent obsession would stay with him for the rest of his life. It was to lead him to his greatest achievement, general relativity, and to his noble failure, unified field theory." [Pai82, p. 140][6]

This is corroborated by many of Einstein's own writings, particularly those from his later years, for instance:

> "I do not consider the main significance of the general theory of relativity to be the prediction of some tiny observable effects, but rather the simplicity of its foundations and its consistency."
> (Einstein, as cited on p. 273 of [Pai82].)

As with Newton and the others, the preference for simplicity is not merely a methodological or heuristical tool, but is taken to correspond to simplicity in the world itself:

> "In my opinion, there is *the* correct path and ... it is in our power to find it. Our experience up to date justifies us in feeling sure that in nature is actualized the ideal of mathematical simplicity."
> (Einstein, as cited on pp. 466–467 of [Pai82].)

Apart from adopting the ontological form of the razor, Einstein and many other scientists also followed its aesthetical interpretation. For instance, Paul Dirac writes:

[6] Notice that Pais seems to identify 'aesthetic arguments' with 'arguments of simplicity'. As far as science is concerned, he is probably to a very large extent in the right. However, in other areas such as art, equating aesthetics and simplicity would be too simple.

> "It is more important to have beauty in one's equations than to have them fit experiment. [...] It seems that if one is working from the point of view of getting beauty in one's equations, and if one has really a sound insight, one is on a sure line of progress."
> ([Dir63, p. 47], as cited on p. 15 of [McA96].)

We may conclude that some of the greatest advances in modern science—Copernicus' heliocentric model, Newton's theory of gravition, and Einstein's theory of relativity—were highly influenced by the value those scientists themselves bestowed on *simplicity* as a virtue of theories and as an indicator of the truth of those theories. In sum, many of the greatest scientists held Occam's Razor in its ontological as well as in its aesthetical form, believing in the simplicity of the world and the truth-indicating properties of beauty.

4.2.3 Applications in Philosophy

Whether or not philosophy is "continuous" with science (and what exactly this might mean) is a debatable issue, which we shall not go into here. One important feature that philosophy shares with science is the importance it places on simplicity as a desirable feature of theories and explanations, and accordingly on Occam's Razor. Particularly in our century, many a philosopher invested a lot of time and effort in Occam's Razor. For instance, Bertrand Russell used the razor so extensively that Passmore calls it "his main philosophical occupation" [Pas67, p. 229]. In this subsection we will mention some applications of Occam's Razor—mainly in its ontological form—in philosophy.

The first and foremost application in philosophy lies in the philosophy of science: Occam's Razor closes the gap between observation and theory. What is this gap, how does it arise, and how can it be closed? The available data—observations, experiments, etc.—constitute the raw material on which a scientific theory is to be built. However, usually more than one theory is consistent with the available data: the data *underdetermine* the theory. That is, usually we can construct many—mutually inconsistent—theories to explain the same set of observational data, and the data do not provide us with further criteria to choose between these various theories. Thus the data leaves open a lot of possible choices. Now Occam's Razor closes this gap by stating that of all theories consistent with the data, we are to choose the simplest one. Accordingly, if we accept Occam's Razor (in some precise form, where 'simplicity' is measurable), there is hardly any underdetermination left![7] Informally, we might even say that the conjunction of the observational data and Occam's Razor *entails* which theory we are to adopt. Thus Occam's Razor may be seen as one of the cornerstones of science.

Our second example concerns *logical positivism*, the logico-scientifically minded movement that sprung from the Wiener Kreis and included people like Schlick, Carnap, and Reichenbach. Following the lead of Wittgenstein's Tractatus 3.328 ("Wird ein Zeichen *nicht gebraucht*, so ist es bedeutungslos. Das ist

[7] The only case where some underdetermination remains, is where *the* simplest theory does not exist, i.e., where several among the consistent theories share the lowest complexity.

der Sinn der Devise Occams.")[8], they attempted to cut away metaphysics by showing its propositions to be meaningless. If some term or proposition has no testable links to our experiental world, they contended, it can only be used in empty babble, but not in any proper, scientifically relevant way. Accordingly, we can do without it, and by Occam's Razor we *should* do without it. Since the logical positivists thought they could show *all* metaphysical terms—such as 'God', 'spirit', 'Ding an sich'—to be without testable empirical content, this allowed them to cleanly shave away all of metaphysics.

Our third example is from the philosophy of mind. Here Occam's Razor is used to argue against *dualism*, the position which says that the physical and the mental are different kinds of "stuff". Are statements like "I am in pain" about something irreducibly mental, or are they actually about material brain processes? A number of materialists—mainly Australian ones, for some reason—have argued for the latter [Pla95, Sma95]. Why? "Mainly because of Occam's Razor" [Sma95, p. 118]. The Australian strategy is to show that materialism, which says that the mental is *identical* to a brain process, is consistent with the facts and much simpler than competing positions like dualism. After all, materialism requires only one kind of stuff, dualism requires two (which, moreover, do not seem to fit together very well). An application of Occam's Razor then suffices to eliminate the competition, leaving materialism as the glorious winner. We can see Occam's Razor at work in the following quote from the notable materialist Jack Smart:

> "If it be agreed that there are no cogent philosophical arguments which force us into accepting dualism, and if the brain process theory and dualism are equally consistent with the facts, then the principles of parsimony and simplicity [= Occam's Razor] seem to me to decide overwhelmingly in favour of the brain-process theory." [Sma95, p. 130].

If Occam's Razor helps us decide between dualism and materialism, it is a very sharp and powerful razor indeed!

A fourth example is in the field of *ontology*. Here *trope theory* has fairly recently been put forward as an alternative to the ancient substance/property ontology [Wil66, Cam90, Bac88]. According to the latter, there are two separate ontological categories: the world consists of things (substances), which have properties. Trope theory, on the other hand, argues that there is only one category, namely *tropes*. A trope is in a way the intersection of a substance and a property: it is a particular property of a particular thing, like the greenness of this particular pea. According to trope theory, tropes are the only ontological category: the world consists of tropes, all tropes, and nothing but tropes. Given that one ontological category is simpler than two, Occam's Razor induces us to favour trope theory over the substance/property ontology.[9]

[8] A similar attitude also appears in the later Wittgenstein, for instance "Hier möchte ich sagen: das Rad gehört nicht zur Maschine, das man drehen kann, ohne daß Anderes sich mitbewegt" [Wit53, §271].

[9] However, it should be noted that this example differs somewhat from the others. Namely,

Our fifth and final example uses Occam's Razor as an argument for atheism, or at least for agnosticism (both atheism and agnosticism would horrify Occam himself!). Basically, the argument is that science is sufficient to explain the phenomena, and hence there is no need to invoke the existence of a God, in some form or other, as an additional explanatory factor. But if we do not need God in our explanations, Occam's Razor tells us that we should not postulate His existence, thus making us either an atheist (if we postulate His non-existence) or an agnostic (if we suspend judgment as to His existence).

As a final comment on the application of Occam's Razor, it should be noted that such razing arguments are not *conclusive*, in the sense that they do not establish something beyond any doubt. In each of the five above examples, Occam's Razor provides *one* argument, *one* good reason for the simpler position, but not a conclusive proof. After all, Occam's Razor cannot be conclusive always, because sometimes the simplest explanation or hypothesis simply is not the right one and turns out to be false later on. Nevertheless, arguing for some position on the basis of Occam's Razor is not futile or empty either, because as we will see, one can establish a positive correlation (though clearly not a perfect one) between simplicity on the one hand, and "truth" or adequacy on the other. Thus, given competing positions, it still makes sense to favour the simpler.

4.3 Occam and PAC Learning

The examples given in the previous section are probably sufficient to convince the reader of the central place Occam's Razor holds in science and philosophy. In this and the following two sections, we will see how we can mathematically formalize and "prove" versions of Occam's Razor. Each of these three versions is closer to the methodological than to the ontological interpretation of the razor, though they also capture the aesthetical interpretation to the extent that simplicity can be equated with beauty. Clearly, in order to formalize Occam's Razor, we need some quantitative measure of simplicity. Not surprisingly, Kolmogorov complexity will serve this role in each of the three settings. The first of these, the topic of the present section, deals with PAC learning.

Occam's Razor states that the simplest consistent hypothesis should be selected. However, in many cases it is computationally rather costly to really find the *simplest* hypothesis, while it is often much easier to find a relatively *simple* hypothesis. For instance, given two finite sets S and T of sentences (known to be grammatical and ungrammatical, respectively), it is not very difficult to find a Deterministic Finite Automaton (or a regular grammar) which generates a language L that contains S and is disjoint from T, whereas finding the *smallest* such DFA is known to be $\mathcal{NP}$-complete, and hence probably not efficiently solvable [GJ79, p. 267]. Accordingly, we will weaken Occam's Razor somewhat to the following:

> Selecting relatively simple hypotheses is a good strategy.

the choice of substance/property versus a trope ontology is more like a choice of conceptual scheme (or language), than a choice between competing empirical hypotheses.

Below we give a theorem which can be seen as a formal counterpart to this version of the razor within the PAC learning framework. Here we will assume some representation R, without explicitly mentioning R each time. Furthermore, we assume R as well as the domain have a binary alphabet $\{0,1\}$, so all names of concepts and all examples are binary strings.

An *Occam algorithm* is a learning algorithm that follows Occam's preferred strategy: it reads examples and outputs a consistent hypothesis that is significantly simpler than those examples. The main result of this section will be that an efficient Occam algorithm is an efficient PAC learning algorithm. Informally this means that if we follow Occam's Razor, then we thereby *automatically* learn probably approximately correctly!

There exist several subtly differing versions of the Occam's Razor theorem. It was originally proved by Blumer, Ehrenfeucht, Haussler, and Warmuth [BEHW87], but see also [AB92, Theorem 6.5.1], [Nat91, Theorem 3.3], [KV94, Theorems 2.1 and 2.2], [LV97, Theorem 5.4.1]. Most of these results measure the simplicity of a hypothesis by the length of the name the learning algorithm outputs. [LV97] is the only one that uses the more sophisticated approach of measuring simplicity by the Kolmogorov complexity of that name. Unfortunately, the PAC-framework used there is somewhat simpler than the one adopted by us (in particular, it takes $\delta = \varepsilon$, and there is no length parameter n). Accordingly, we will have to prove a version of our own, adapting and combining some definitions and proofs that can be found in the literature.

Formally, an Occam algorithm is defined as an algorithm that, when given a set of examples, outputs the name of a concept (consistent with those examples) with "small" Kolmogorov complexity:

Definition 4.1 Let $\mathcal{F}$ be a concept class. A learning algorithm L for $\mathcal{F}$ is called an *Occam algorithm* if there exist constants $0 \leq \alpha < 1$ and $\beta \geq 1$ such that, whenever L is given a set S of m examples for a target concept f, L outputs a name r of a concept g satisfying

- g is consistent with S.
- $K(r) \leq (mn)^{\alpha} l_{min}(f)^{\beta}$, where n is the length parameter.

Moreover, L is a *polynomial-time* Occam algorithm if also

- There is a polynomial $p(n,m)$ such that, given length parameter n and m examples, L needs at most $p(n,m)$ steps.

$\diamond$

The interesting and perhaps puzzling part of this definition is the condition $K(r) \leq (mn)^{\alpha} l_{min}(f)^{\beta}$. It says that the Kolmogorov complexity of the output should be bounded by a polynomial in mn (the number of examples times the maximal length of examples) and $l_{min}(f)$ (the shortest name of the target concept f). If we have a set S of m examples, each of length at most n bits, then we can often simply construct a consistent hypothesis by recording the given examples and their labels explicitly, and assigning all other, unseen elements

of the domain the label 0. Since each example is at most n bits, recording m examples and their labels takes roughly mn bits. This concept represents the examples explicitly, but does not really "learn" anything. However, because $\alpha < 1$ and $l_{min}(f)^\beta$ and n are fixed, for sufficiently large m we get $(mn)^\alpha l_{min}(f)^\beta < mn$, so an Occam algorithm will have to find a g which is simpler than the concept that simply records the given examples. Accordingly, for sufficiently large m an Occam algorithm will have to *compress* the data: the output concept should be simpler than the examples themselves.

The following theorem shows that Occam's Razor really works, in the sense that an Occam algorithm can meet the requirements of a PAC algorithm. In order not to impair readability, we have deferred the very technical proof to the appendix of this chapter.

Theorem 4.1 *Let $\mathcal{F}$ be a concept class. If there is a polynomial-time Occam algorithm for $\mathcal{F}$, then $\mathcal{F}$ is polynomial-time PAC learnable.*

Cutting through the above notation, the result says the following: if we can follow Occam's Razor in the sense of having an efficient algorithm that selects short hypotheses, then this algorithm is automatically guaranteed to learn probably approximately correctly. In less words: efficiently selecting short hypotheses is a good strategy, and we can formally prove this in the PAC framework!

It is important to note carefully what has and has not been proved here. What *has* been proved, is that *if* we can follow Occam's Razor in the sense of having an efficient Occam algorithm, then this algorithm will be a "good" (i.e., PAC) hypothesis selector. What has *not* been proved, is that we can always follow Occam's Razor. One can easily think of cases (for instance, if the domain X consists of all finite binary strings, and as concept class we have $\mathcal{F} = 2^X$) where a sufficiently long given sequence of m examples, each of length n, has a Kolmogorov complexity of about mn. In such cases, compression of the examples to within the bound $K(r) \leq (mn)^\alpha l_{min}(f)^\beta < mn$ will not always be possible; no Occam algorithm will exist in this case. In sum: we can prove that following Occam's Razor is a good thing, but in some cases it may not be possible to implement the razor.[10]

4.4 Occam and Minimum Description Length

Of the three formalizations of Occam's Razor discussed in this chapter, the *Minimum Description Length* (MDL) principle probably comes closest to our informal understanding of Occam's Razor. It was invented by Jorma Rissanen [Ris78, Ris89], motivated by Ray Solomonoff's work (to be described in the next section) and tells us to select the theory which most compresses the data:

[10] There actually exist some weak converses to related forms of this theorem (see [KV94, Exercises 2.3 and 4.2] and [Nat91, Theorem 3.4]), which show that, under certain conditions, classes that are efficiently learnable are also learnable using Occam algorithms. Unfortunately, we have not been able to prove an exact converse to our present version of the theorem.

> Given a sample of data and an effective enumeration of the appropriate alternative theories, the best theory is the one that minimizes the sum of
>
> - the length (in bits) of the description of the theory;
> - the length (in bits) of the data when encoded with the help of the theory.

The idea here is that we should balance between putting too much and too few detail in the theory. In the most likely cases, the data contain a more or less regular pattern with some additional noise (irregularities due to measuring errors, etc.). If we tried to include the noise in the theory, the description of the theory would get rather large and detailed, while the length of the data encoded with the help of the theory would get smaller, since the data can then already be reconstructed largely from the theory itself. However, in this case we would probably *overfit* the data, i.e., we would lose predictive power by focussing too much on the accidental noisy details of the given data. On the other hand, we can minimize the length of the theory by including *all* the details of the data in the second part. But in this case, we would have a virtually empty theory and, again, no predictive power. The right approach lies somewhere in the middle, and achieves predictive power by balancing the descriptions of the theory and data by minimizing the *sum* of their lengths.

To the extent that Occam's Razor is intuitively acceptable, the MDL principle seems acceptable as well. In fact, however, under certain conditions it can be *proved* that MDL does indeed work well. Suppose we want to explain some data D. Let $\mathcal{H} = \{H_1, H_2, \ldots\}$ be the (finite or countably infinite) set of all possible hypotheses or theories. We assume each possible D and each H_i is somehow encoded as a binary string. These H_i are assumed to be exhaustive as well as mutually exclusive. Furthermore, we assume each H_i induces a probability distribution $\Pr(\cdot|H_i)$ on the possible data D, so it makes sense to speak of the conditional probability $\Pr(D|H_i)$ that D arises, given that H_i is true. We will asume each $\Pr(\cdot|H_i)$ to be recursive. For a recursive probability distribution P, we define $K(P)$ to be the length of a shortest Turing machine that computes $P(x)$ given x, for all x. Let $\mathbf{P}$ be an *a priori* probability distribution on the possible hypotheses ($\sum_{H\in\mathcal{H}} \mathbf{P}(H) = 1$). $\mathbf{P}(H_i)$ can be interpreted as the probability we are willing to confer upon H_i *before having seen any data*. $\mathbf{P}$ can be used to define an a priori probability on the possible D: $\Pr(D) = \sum_{H\in\mathcal{H}} \Pr(D|H)\mathbf{P}(H)$. This is the sum, over all H, of the probability that H is true and causes D. If both Pr and $\mathbf{P}$ are computable, then $\Pr(D)$ can be computed (or approximated if $\mathcal{H}$ is infinite).

So, we have a prior distribution on the hypotheses and on the data. Now we observe the data D, which induces us to adjust the probabilities we confer upon the different hypotheses. For example, if $\Pr(D|H_i) = 0$, then observing D effectively rules out the possibility that H_i is the right hypothesis, and we can put the probability $\Pr(H_i|D) = 0$: given D, we know H_i is impossible. Similarly we can update the other probabilities, incorporating the fact that D was observed. *Bayes's Theorem* or *Bayes's Rule* (which is actually due to Laplace rather than Bayes) is basically a rule which tells us how to update the

a priori probability $\mathbf{P}(H)$ to an *a posteriori* probability $\Pr(H|D)$:

$$\Pr(H|D) = \frac{\Pr(D|H)\mathbf{P}(H)}{\Pr(D)}.$$

This rule gives a mathematically sound way to update the probabilities conferred upon the hypotheses, given that D has been observed. The best hypothesis H is the most probable one, given D, i.e., the H that maximizes $\Pr(H|D)$ (there may actually be several different such H, so we should rather speak of *a* best hypothesis rather than *the* best hypothesis).[11]

This approach gives a sound and objective way to select an optimal hypothesis, and if we have $\mathbf{P}$ (as well as the resources to do all required computation) then this approach is the best way to select a hypothesis. Unfortunately, the problem is that we usually will *not* know $\mathbf{P}$. Thus the Bayesian approach is an optimal but usually unworkable hypothesis selection scheme.

However, using Kolmogorov complexity we can show that the MDL principle is often a very good *approximation* to the above Bayesian approach. First, if we take negative logarithms on both sides of Bayes's Rule, we get the equivalent

$$-\log\Pr(H|D) = -\log\mathbf{P}(H) - \log\Pr(D|H) + \log\Pr(D). \tag{4.1}$$

Since $\log\Pr(D)$ is fixed independent of H, choosing a hypothesis H which *maximizes* $\Pr(H|D)$ is equivalent to choosing an H which *minimizes* $-\log\mathbf{P}(H) - \log\Pr(D|H)$. Thus hypothesis selection according to Bayes's Rule is equivalent to selecting a hypothesis H that minimizes $-\log\mathbf{P}(H) - \log\Pr(D|H)$. Of course, the same old problem still obtains: we do not know $\mathbf{P}(H)$.

Now suppose we replace this unknown prior $\mathbf{P}$ by the following *universal distribution*:

$$\mathbf{m}(x) = 2^{-K(x)}.$$

This distribution assigns each natural number or binary string x a non-zero probability.[12] The universal distribution incorporates Occam's Razor by giving simple hypotheses high probability: if $K(H)$ is small, then $\mathbf{m}(H)$ will be relatively high. This $\mathbf{m}$ is enumerable, but not recursive. It has the property that for every enumerable probability distribution P (in particular, for every *recursive* P), there is a constant c such that for every x we have $c\mathbf{m}(x) \geq P(x)$.

What happens if we replace $\mathbf{P}$ by $\mathbf{m}$? Let us first impose three conditions: (1) we restrict attention to *recursive* $\mathbf{P}$, (2) the "true" hypothesis H is $\mathbf{P}$-random, and (3) the data D is $\Pr(\cdot|H)$-random. The latter two conditions

[11] An alternative is to use the *Maximum Likelihood* principle, which selects a H with highest $\Pr(D|H)$. In other words, this principle favours a hypothesis that makes the data most probable. However, this principle may give strongly counterintuitive results. For instance, the principle may well select D as its rather useless hypothesis, because $\Pr(D|D) = 1$ is maximal.

[12] It can be shown that $\sum_x \mathbf{m}(x) < 1$ [LV97, Lemma 4.3.2], so $\mathbf{m}$ is not quite a probability distribution. However, we can construct an additional dummy object u and define $\mathbf{m}(u) = 1 - \sum_x \mathbf{m}(x)$ in order to make probabilities sum to one.

It should be noted that we attach no "objective" character to the universal prior probability; we do not consider it the "true" probability of theories (it is not even clear what this might mean). Rather, we take an instrumentalist stance, satisfying ourselves with the conclusion that using $\mathbf{m}$ as a prior gives good results.

informally mean that H is a "typical" hypothesis for $\mathbf{P}$, and D is "typical" data for H. Since an overwhelming majority of all strings will be "typical" in this sense, this is not a very strong restriction.[13] Under these conditions it can be shown that the following inequality holds [LV97, p. 357]:

$$K(H) - K(P) \leq -\log \mathbf{P}(H) \leq K(H). \tag{4.2}$$

This means that $-\log \mathbf{P}(H)$ and $-\log \mathbf{m}(H) = K(H)$ are equal to within a fixed constant (namely $K(P)$), which is independent of H.

Similarly, if we define $K(x|y)$ as the length of a shortest Turing machine that generates x, *given string y on a special input tape*, then, under the above conditions, $-\log \Pr(D|H)$ and $K(D|H)$ are approximately equal:

$$K(D|H) - K(\Pr(\cdot|H)) \leq -\log \Pr(D|H) \leq K(D|H). \tag{4.3}$$

As we saw, our original aim to maximize $\Pr(H|D)$ is equivalent to minimizing $-\log \mathbf{P}(H) - \log \Pr(D|H)$. Since $K(H)$ approximates $-\log \mathbf{P}(H)$ and $K(D|H)$ approximates $-\log \Pr(D|H)$, minimizing $-\log \mathbf{P}(H) - \log \Pr(D|H)$ now turns out to be approximately equivalent to minimizing $K(H) + K(D|H)$. That is, we want to select a H which minimizes the description length of H and the description length of the data D, encoded with the help of H. But this is just what the MDL principle says! Therefore the Bayesian approach, which is optimal but infeasible because we do not know the prior $\mathbf{P}$, can be approximated using Kolmogorov complexity, which gives us the MDL principle.

How good an approximation is MDL to the Bayesian selection scheme? If we define $\alpha(P, H) = K(\Pr(\cdot|H)) + K(P)$ and add up inqualities 4.2 and 4.3, we obtain the following:

$$K(H) + K(D|H) - \alpha(P, H) \leq -\log \mathbf{P}(H) - \log \Pr(D|H) \leq K(H) + K(D|H).$$

We call an H *admissible* if it satisfies this inequality. As we can see, if this inequality holds and $\alpha(P, H)$ is small, then $K(H) + K(D|H)$ and $-\log \mathbf{P}(H) - \log \Pr(D|H)$ will be close to each other, and the MDL principle will provide a good approximation to Bayesian hypothesis selection. The following is Theorem 5.5.1 of [LV97]:

Theorem 4.2 *Let $\alpha(P, H)$ be small. Then Bayes's Rule and MDL are optimized (or almost optimized) by the same hypothesis among the admissible H's. That is, there is one admissible H that simultaneously (almost) minimizes both $-\log \mathbf{P}(H) - \log \Pr(D|H)$ (selection according to Bayes's Rule) and $K(H) + K(D|H)$ (selection according to MDL).*

[13] It is not a very surprising requirement that the data should be random or typical in order for MDL to work. After all, non-typical data may well be very misleading. For example, suppose we are given some data pertaining to the movements of the planets (a number of points in space and time). Since the planetary orbits are not circular, typical data will be inconsistent with the theory that planets move in perfect circles around the sun. However, if we are given non-typical data which *are* consistent with this theory, then we cannot be blamed for jumping to the simple but wrong conclusion that planets move in circles. Analogously, MDL's preference for simple theories may go wrong if the given data is non-typical and just happens to be consistent with one or more very simple but wrong theories. MDL would select an overly simple and incorrect theory, but it cannot be blamed for this: if the right information just isn't there, MDL can't be expected to extract it from the data.

Thus Occam's Razor in the form of the MDL principle can be justified if the true hypothesis and the data are typical. Notice, however, that the non-computability of Kolmogorov complexity makes MDL in this form still beyond algorithmic means. The problem now is no longer that we do not know the prior $\mathbf{P}$, but that we generally cannot compute the description lengths $K(H)$ and $K(D|H)$. Therefore most "real-world" versions of MDL restrict the domain of application such that description lengths are (efficiently) computable. Nevertheless, despite the fact that Occam's Razor in its MDL-form is not fully implementable, the above results do vindicate the razor by showing that *if* we somehow follow the razor, we will probably get good results.

4.5 Occam and Universal Prediction

In this section we will not be concerned with choosing a good theory or hypothesis, but with *predicting* the future from the past. Here we will describe Ray Solomonoff's prediction procedure, for which Kolmogorov complexity was initially introduced [Sol64].[14]

First, let us mention explicitly that optimal prediction does not coincide with prediction according to the best hypothesis H. In fact, sometimes prediction can yield better results if we refrain from explicitly selecting one hypothesis among the many possibilities. Consider an unfair coin which has an unknown chance p of coming up 'heads'. Suppose there are only two hypotheses possible, H_1 says that $p = p_1 = 1/3$ and H_2 says $p = p_2 = 2/3$, and we have determined that the probability of H_1 being true is $2/3$, the probability of H_2 is $1/3$. Then we clearly should select H_1 as the best (most likely) hypothesis. Yet the best prediction for p, given our knowledge, is $2/3 \cdot p_1 + 1/3 \cdot p_2 = 4/9$. If we were to bet against this coin, then the prediction $p = 4/9$ would be more profitable than the H_1-prediction $p = 1/3$. Thus the best hypothesis need not give the best prediction.

The formal setting in which the prediction will take place is simple, abstract, and austere. We are given a finite initial segment x of a binary sequence, and our aim is to predict how the sequence will continue. Intuitively, we can think of the given initial segment x as a description of relevant aspects of the past, and our predicting the continuation of the sequence is like predicting the future. Where does this x come from? Informally, it may be seen as the outcome of past observations or experiments. Formally, we assume it is drawn according to some unknown *semimeasure* μ on the set of infinite binary sequences.

What is a semimeasure? Recall from Section 3.4.2 that a *measure* on $\{0,1\}^\infty$ (the set of infinite binary sequences) is a function μ, from $\{0,1\}^*$ to $[0,1]$, such that:

$$\mu(\epsilon) = 1.$$
$$\mu(x) = \mu(x0) + \mu(x1).$$

[14] An interesting historical point for the relation between philosophy and Kolmogorov-complexity-based prediction: during his physics studies at the University of Chicago in the late 1940s, Solomonoff followed a course given by Rudolf Carnap, who was at that time very active in research on induction and prior probabilities.

Here $\mu(x)$ is interpreted as the probability that a string from $\{0,1\}^\infty$ starts with x. A *semimeasure* needs to satisfy only the following weaker conditions:

$$\mu(\epsilon) \leq 1.$$
$$\mu(x) \geq \mu(x0) + \mu(x1).$$

Clearly, a measure is a semimeasure, but not always vice versa.[15]

As mentioned, our given x is a description of (parts of) the world, and we assume this world to behave in accordance with some unknown semimeasure μ, which assigns a number $\mu(x)$ to every finite binary string x. Given μ, we can define the *conditional* semimeasure $\mu(y|x)$ as

$$\mu(y|x) = \frac{\mu(xy)}{\mu(x)}.$$

Intuitively, $\mu(y|x)$ is the "probability" that, given an initial segment x, the sequence will continue with y (it's not quite a probability, because the probabilities need not sum to one in a semimeasure and need to be renormalized, but it is easiest still to think of $\mu(x)$ and $\mu(y|x)$ as probabilities). So, for example, if $\mu(1010) = 0.5$ and $\mu(10100) = 0.2$, then $\mu(0|1010) = 0.2/0.5 = 0.4$. If, moreover, $\mu(1|1010) = 0.1$, then we can interpret this as an 80% chance that the next bit will be a 0 and a 20% chance that it will be 1, given 1010 as initial segment. Note that this setting is able to incorporate a *non-deterministic* world.

Given x, we want to predict how the sequence continues, in such a way that our predictions do not diverge too far from the true but unknown distribution μ. Let us restrict attention to predicting only the next bit of the sequence (further bits can then be predicted by reiterating the procedure). As our world may be probabilistic (non-deterministic), it is best if our predictions are probabilistic as well: rather than definitely predicting "the next bit will be a 0", we should make predictions of the form "with probability 0.6, the next bit will be a 0". How can we make a good prediction in a uniform way? In general, we can't: if we allow arbitrary semimeasures μ as our "world", then basically anything can happen, and there is no prediction method that will work well universally. However, now suppose we restrict attention to *recursive* μ's, that is, to μ's for which there is an effective procedure to calculate $\mu(x)$ for every x. This is not a very strong restriction; for example, all usual distributions one finds in books on statistics (the normal one, the uniform one, the exponential one, etc.) are recursive.

Now, almost miraculously, under this mild restriction we can use a *single* semimeasure, the *universal* semimeasure $\mathbf{M}$, to predict how the sequence x will continue, *no matter what the actual μ is!* Informally, this universal $\mathbf{M}$ in a way "combines" all enumerable semimeasures, weighed according to their Kolmogorov complexity. We know what the Kolmogorov complexity of a binary

[15] The fact that probabilities need not sum to one in a semimeasure is a technical convenience. One can always "renormalize" a semimeasure to a measure. For example, if we are given x as initial segment, and we have $\mu(x0) = 0.3$ and $\mu(x1) = 0.2$, then we can use $P(x0) = 0.6$ and $P(x1) = 0.4$ as probabilities, which sum to one. However, in general the measure obtained by renormalizing an enumerable semimeasure need not be enumerable itself (an example is $\mathbf{M}$, defined below).

string is, but what is the Kolmogorov complexity of an enumerable semimeasure? One can effectively enumerate all enumerable semimeasures—more precisely, the Turing machines that enumerate them—in a sequence $\mu_1, \mu_2, \mu_3, \ldots$ (see the proof of Theorem 4.3.1 of [LV97]). Given a particular enumeration like this, we can define $K(\mu_i) = K(i)$: the complexity of an enumerable semimeasure is the complexity of its index in the enumeration (after all, given a Turing machine that constructs the enumeration, all we need in order to be able to construct μ_i is its index i). Now $\mathbf{M}$ is defined as the weighed sum of all these μ_i:

$$\mathbf{M}(x) = \sum_{n=1}^{\infty} 2^{-K(\mu_n)} \mu_n(x).$$

Each μ_i is a possibility for the true μ (the one from which x has been drawn), and hence may be seen as a possible *hypothesis*. The above definition of $\mathbf{M}$ incorporates Occam's Razor in giving preference to simple hypotheses: the simple ones (the ones with low $K(\mu_i)$) are considered more preferable, and are assigned high weight $2^{-K(\mu_i)}$ accordingly. The universal $\mathbf{M}$ is an enumerable semimeasure, but it is not recursive. 'Universal' here means that $\mathbf{M}$ *multiplicatively dominates* all enumerable semimeasures: for every such semimeasure μ_i, there is a constant c_i such that $c_i\mathbf{M}(x) \geq \mu(x)$, for every x.[16] Namely, if we put $c_i = 2^{K(\mu_i)}$, then $c_i\mathbf{M}(x) = c_i \sum_{n=1}^{\infty} 2^{-K(\mu_n)}\mu_n(x) \geq c_i 2^{-K(\mu_i)}\mu_i(x) = \mu_i(x)$, for every x. (Actually, the particular $\mathbf{M}$ that we defined here is just one example of a universal distribution; there are others with the same property.)

To repeat our prediction problem once more: we get an initial segment x, and we want to predict the next bit, which is either 0 or 1. Let us look at predicting the probability of getting a 0 as next bit. The true prediction would of course be $\mu(0|x)$. But, alas, μ is not known, and there would not be much to learn if it were. What happens if we use $\mathbf{M}(0|x)(= \mathbf{M}(x0)/\mathbf{M}(x))$ as our prediction? Surprisingly, it can be shown that this gives a very good prediction indeed, still assuming μ to be recursive. How good a prediction is $\mathbf{M}(0|x)$, compared to $\mu(0|x)$? Suppose our initial segment x has length $n-1$, and we want to predict the nth bit. The following S_n measures the sum of the errors for all possible x of length $n-1$ (we square the errors in order to avoid positive and negative errors to cancel out each other):

Definition 4.2 S_n is the μ-expected square of the difference in μ-probability and $\mathbf{M}$-probability of 0 occurring at the nth prediction:

$$S_n = \sum_{l(x)=n-1} \mu(x)(\mathbf{M}(0|x) - \mu(0|x))^2.$$

◇

The following result [LV97, Theorem 5.2.1] tells us that $\mathbf{M}$ is a good predictive tool for *any* recursive μ.

[16] This is analogous to the universal distribution $\mathbf{m}$ of the last section. However, note the distinction between $\mathbf{m}$ and $\mathbf{M}$: $\mathbf{m}$ assigns a probability to finite strings, while $\mathbf{M}$ assigns a probability to finite prefixes of infinite strings.

Theorem 4.3 *If μ is a recursive semimeasure, then $\sum_{n=1}^{\infty} S_n \leq 0.5 \ln 2 \cdot K(\mu) (\approx 0.35 K(\mu))$.*

This means not only that the error S_n converges to 0, but also that it does so rather fast: S_n must converge to 0 faster than $1/n$ does, in order for $\sum_{n=1}^{\infty} S_n \leq 0.5 \ln 2 \cdot K(\mu)$ to hold.[17]

It is instructive to see exactly in which way Occam's Razor is justified by the above results. These results do not *prove* a version of Occam's Razor; rather, we *presuppose* Occam's Razor (by weighing the different μ_i according to their complexity in the construction of $\mathbf{M}$) and show that it has beneficial consequences to do so, namely a provably successful induction procedure. Let us stress again, however, that such an abstract justification of Occam's Razor does not imply its efficient applicability. In particular, the universal measure $\mathbf{M}$, though enumerable, is not recursive, and getting a good approximation of its value can be very expensive computationally.

4.6 On the Very Possibility of Science

Science aims at describing the multitude of observations and data in simple and elegant theories. In other words, science aims at *compression*. As we have seen in the last three sections, that compression is a good strategy can even be justified mathematically. However, in order for science to be possible in the first place, compression should be possible—if there is nothing we can compress, there is nothing to learn, and everything will just be an incomprehensible flurry.

Let us consider the possibility that there is nothing we can compress, so nothing that science can successfully work on. Suppose the long binary string x is a complete description (in some sense) of our universe. We can roughly distinguish two cases: (1) x is significantly compressible (i.e., $K(x)$ is much smaller than the length of x), or (2) it is not. In the first case, there is clearly a structure inherent in x that scientific research can latch on to. But what about the second case: if x is completely random, how then is science possible? Fortunately, in this case it can be shown that the irregular x will probably contain some very regular non-random substrings! ([LV97, Section 2.6] contains some results to this effect.) This result is not as surprising as it may seem at first sight. After all, if we flip a fair coin a huge number of times, then the resulting sequence of 'heads'/'coins' will probably be completely random; but still a long sequence of 'heads' is bound to come up at some point, simply because of the laws of probability. So in this second case, even though "the world as a whole" (i.e., x) is random and a successful "theory of everything" would be beyond us, some parts of the world (substrings of x) will be non-random and will enable fruitful scientific research to take place. In either case, whether x is compressible or not, we can rest (?) assured: it is reasonable to expect that science is at least possible in *some* domains or parts of the world.

Accordingly, scientific research is concerned, first, with identifying *which* substrings (i.e., which parts of the world) are sufficiently regular to enable

[17] This is so because $\sum_{n=1}^{\infty} 1/n = \infty > 0.5 \ln 2 \cdot K(\mu)$.

fruitful research, and, second, with identifying what exactly the structure in those regular substrings is. Clearly, whether a particular science can be successful depends on the regularity of its subject matter; it seems fair to say that the substrings of our world that successful sciences like physics are working on, are much more regular than the substrings that are the object of, for instance, sociology.

4.7 Summary

Occam's Razor tells us that we should prefer simpler theories over more complex ones; a prescription which is generally followed in science as well as philosophy. What justifies this razor? We described three different formal settings in which a form of Occam's razor could be mathematically justified. Firstly, in the PAC framework, an Occam algorithm is an algorithm that outputs names of concepts with "small" Kolmogorov complexity, compared to the given examples and the target concept. An efficient Occam algorithm automatically learns probably approximately correct concepts. Secondly, the Minimum Description Length principle tells us to select a hypothesis such that the Kolmogorov complexity of hypothesis + examples is minimized. It can be shown that in "typical" cases, this approach does indeed select an approximately optimal hypothesis. Thirdly, Solomonoff's prediction procedure predicts the continuation of binary sequences using a combination of all possible (computable) probability measures, weighed according to their complexity. Such predictions quickly converge to the true values.

What these three approaches have in common, is that they show that simplicity is to be favoured, and hence *compression* is a good thing in science. In the last section, we saw how the theory of Kolmogorov complexity renders it very likely that compression—and hence successful scientific activity—is at least possible in *some* domains.

4.A Proof of Occam's Razor (PAC Version)

In this appendix we give the proof of Theorem 4.1 from Section 4.3, which stated a version of Occam's Razor in the framework of PAC learning. Except for using a Kolmogorov complexity bound rather than a simple length bound on the names the algorithm outputs, this proof is analogous to the proof of [AB92, Theorem 6.5.1].

Theorem 4.1 Let $\mathcal{F}$ be a concept class. If there is a polynomial-time Occam algorithm for $\mathcal{F}$, then $\mathcal{F}$ is polynomial-time PAC learnable.

Proof Let L be a polynomial-time Occam algorithm for $\mathcal{F}$. Consider a target concept $f \in \mathcal{F}$ and a distribution $\mathbf{P}$ on the domain. L reads a set S of m examples for f, and outputs a name r of a concept $g \in \mathcal{F}$, consistent with S, satisfying $K(r) \leq (mn)^\alpha l_{min}(f)^\beta$. We will see how we can choose m in such a (polynomially-bounded) way that the requirements of a PAC algorithm are satisfied. In the following, we use 'l' to abbreviate '$l_{min}(f)$'.

First note that the number of binary strings of length at most $(mn)^\alpha l^\beta$ is

$$2^0 + 2^1 + 2^2 + \ldots + 2^{(mn)^\alpha l^\beta} = 2^{(mn)^\alpha l^\beta + 1} - 1.$$

L can only output a name r for a concept g if $K(r) \leq (mn)^\alpha l^\beta$. Hence the number C of concepts for which L can output a name satisfies $C < 2^{(mn)^\alpha l^\beta + 1}$.

Let us call a concept $g \in \mathcal{F}$ *bad* if it has too large an error: $\mathbf{P}(f \Delta g) > \varepsilon$. The probability that some particular bad concept g is consistent with one example for f is $1 - \mathbf{P}(f \Delta g) < 1 - \varepsilon$; hence the probability that g is consistent with each of our m examples is at most $(1-\varepsilon)^m$. Now, the probability that *at least one* of the C possible output concepts of L is bad, is at most

$$C(1-\varepsilon)^m < 2^{l^\beta (mn)^\alpha + 1}(1-\varepsilon)^m.$$

If we can bound this probability by δ, then with probability at least $1-\delta$, L will output a good (i.e., non-bad) concept, thus satisfying the requirements of a PAC algorithm. Accordingly, we want to choose m such that:

$$2^{(mn)^\alpha l^\beta + 1}(1-\varepsilon)^m < \delta.$$

Equivalently, taking natural logarithms on both sides:

$$(mn)^\alpha l^\beta \ln 2 + \ln 2 + \ln(1-\varepsilon)^m < \ln \delta.$$

If we abbreviate $A = n^\alpha l^\beta \ln 2$ and $B = \ln(2/\delta)$, we can rewrite this to:

$$Am^\alpha + B < -\ln(1-\varepsilon)^m = -m \ln(1-\varepsilon).$$

Because $\varepsilon < -\ln(1-\varepsilon)$, the above inequality is implied by the following:

$$Am^\alpha + B < m\varepsilon.$$

Dividing by $m^\alpha \varepsilon$ yields:

$$\frac{A + B/m^\alpha}{\varepsilon} < m^{1-\alpha}.$$

Since $B/m^\alpha \leq B$, the above inequality is implied by:

$$\frac{A + B}{\varepsilon} < m^{1-\alpha}.$$

This inequality holds if we choose

$$m > \left(\frac{A + B}{\varepsilon}\right)^{1/(1-\alpha)}.$$

Thus, choosing m in this way, L will output a name of a concept g such that with probability at least $1 - \delta$, we have $\mathbf{P}(f \Delta g) \leq \varepsilon$.

It remains to check the efficiency of L. Since L is an Occam algorithm, it runs in time polynomial in m and n. Furthermore, it is easy to see that our choice of m can be bounded from above by a polynomial in $1/\delta$, $1/\varepsilon$, n, and l. It follows that L runs in time polynomial in $1/\varepsilon$, $1/\delta$, n, and l, in accordance with Definition 1.12. Hence $\mathcal{F}$ is polynomial-time PAC learnable. □

Chapter 5

Summary and Conclusion

Though each of the previous chapters had its own short summary, it might benefit the reader's overview of the thesis if we wrap things up once more. Accordingly, in the following pages we will take stock, summarizing in non-technical language what we have done.

5.1 Computational Learning Theory

Chapter 1 introduced *computational learning theory*, the branch of Artificial Intelligence which studies the complexity and other theoretical properties of mechanisms for learning. Given some examples—positive and negative instances of some unknown target concept (a subset of a domain of objects)—our aim is to learn this concept. However, since the examples will usually not give complete information about the target, we cannot expect to learn this target perfectly. Instead, we can only hope to learn an *approximately correct* concept: a concept which diverges only slightly from the target, in the sense that the target and the learned concept will agree on most members of the domain. Moreover, since the given set of examples may be biased and need not always be a good representative of the target concept as a whole, we cannot even expect to learn approximately correctly every time. Accordingly, the best we can do, is learn a concept which is *probably* approximately correct (PAC).

The model of PAC learning formalizes this idea in precise mathematical terms. In this model, a class of concepts is considered to be learnable[1], if there exists an algorithm which efficiently learns a PAC concept whenever the target is drawn from that class. This can both be seen as a rough model of learning by children or human beings generally, and as a model of theory construction in empirical science.

The PAC model can be extended in various ways. Firstly, we may allow the learning algorithm access to an *oracle*. The oracle can answer certain questions posed by the learner, for instance membership queries (which ask whether a certain object is a member of the target), or equivalence queries (which ask whether a certain concept is equal to the target). In the latter case, the PAC

[1] Polynomial-time PAC learnable or polynomial-time PAC predictable.

requirement is usually strengthened to the requirement that the target is identified exactly by the learning algorithm. Learnability in this stronger model implies learnability in the PAC model. Secondly, we can make the PAC model more realistic by allowing the examples to contain some *noise* (errors), and examining learnability in the presence of such noise.

5.2 Language Learning

Since the late 1950s, the work of Noam Chomsky has dominated linguistics, and has revived the old debate about innate knowledge. Specifically, Chomsky argues that the speed and accurateness with which children learn their native language—despite the poverty of the "input sentences", the examples they receive from parents and others—can only be explained by postulating that the child is already born with some linguistic bias, some pre-knowledge of its native language. Without such a bias, natural languages would not be learnable. Chomsky particularly focuses on *syntax*, identifying a hierarchy of four classes of languages, Type 3 to Type 0, with increasing syntactical complexity. Each of these classes properly includes the simpler ones. The class of Type 0 languages contains all languages that can be enumerated by an algorithm, and hence is the broadest class that lies within the reach of algorithmic means.

In Chapter 2, we looked at learnability results pertaining to such classes of languages, and the results were rather negative. The class of Type 3 languages is not learnable in the PAC model (without membership queries), and neither are the more complex classes. We defined an even simpler class, the class of Type 4 languages, which also turned out to require too much examples to be efficiently learnable. The Type 3 class *is* learnable given the ability to make membership queries, and the Type 2 class may be learnable in this case (this is not known to the author). However, these classes are still too restricted to contain natural languages such as English and Dutch. Finally, the Type 1 and Type 0 classes are not even learnable *with* membership queries.

We can see from ordinary children that the class of natural languages is learnable in some non-technical sense. Assuming the PAC model to be sufficiently realistic, we conjectured that this class is also learnable in the technical PAC sense. From this it follows that the class of languages that children can learn cannot be some broad class such as the Type 1 or Type 0 languages. Therefore, in order to be able to explain how language learning can take place, we must conclude that the class of natural languages is very restricted, and a child must somehow have pre-knowledge of the particular restrictions. Thus indeed we must have a linguistic bias in favour of languages of some specific kind, as Chomsky argued. Whether this bias is *innate* cannot conclusively be proved using learnability arguments, but does seem to be the most likely option.

5.3 Kolmogorov Complexity

Chapter 3 discussed *Kolmogorov complexity* and some of its philosophically interesting consequences and applications. Technicalities apart, the Kolmogorov

complexity $K(x)$ of a finite binary string x is the length of a shortest Turing machine which produces that string, when fed into some universal Turing machine U. The choice of U can affect the value of $K(x)$ only up to a constant that does not depend on x, which makes $K(x)$ a sufficiently objective measure of the complexity of x. The function K is not computable, but it can be algorithmically approximated. For longer strings it converges to Shannon's measure of information content.

Kolmogorov complexity can be used to give an objective definition of the degree of *simplicity* of descriptions (data, theories, etc., represented as binary strings): a description is simple to the extent that it has low Kolmogorov complexity. Because of the great importance of the notion of simplicity—especially in Occam's Razor, see next section—this is a significant formalization. Particularly as a property of scientific theories, simplicity is strongly correlated with the subjective notions of elegance and beauty.

Secondly, Kolmogorov complexity can be used in a formalization of the property of randomness of finite or infinite binary strings with respect to some probability distribution $\mathbf{P}$. Roughly, a string is $\mathbf{P}$-random if it possesses all properties one can attribute to random strings, i.e., if it passes all effective *tests* for randomness. The set of all tests for $\mathbf{P}$-randomness can be combined in a single universal test. In case $\mathbf{P}$ is the uniform measure, which distributes probability uniformly, randomness varies with incompressibility: a finite string is random to the extent that it is incompressible (x is incompressible if $K(x)$ is near to the length of x), and an infinite string is random if each of its prefixes is incompressible. One important example of a random infinite string is the binary expansion of the number Ω, which is the probability that a randomly drawn binary string encodes a *halting* Turing machine. The first n bits of this number contain sufficient information to find out whether any Turing machine of length at most n halts, and hence to find out the answers to the questions that can be encoded in such machines.

Finally, we can use Kolmogorov complexity to prove Gödel's important incompleteness theorem, without invoking the self-referring constructs employed in the usual proof. Basically, the proof shows that any finite (or recursively enumerable) set of axioms will have only a finite complexity, and hence will not contain sufficient "information" to prove all truths about the infinite incompressible string Ω. Hence it is not possible to capture all mathematical truths in a formal, axiomatic theory.

5.4 Occam's Razor

Chapter 4 dealt with mathematical justifications of Occam's Razor. In its most often cited form—which cannot be found in Occam's writings—this says that "entities are not to be multiplied without necessity". More broadly, we can render Occam's Razor as saying that we should always prefer the *simplest* hypothesis or theory among those that are consistent with the data. This principle can be interpreted in at least three ways: as merely a principle of method; ontologically ("Selecting simple theories is good because the world itself is rela-

tively simple"); or aesthetically ("Beauty—of which simplicity is an important aspect—indicates truth"). The preference for simplicity profoundly influences science and philosophy, but is usually accepted without further justification.

The problem in formally justifying Occam's Razor lies in the notion of simplicity. If we solve this by identifying simplicity with low Kolmogorov complexity, we can give formal justifications of the razor in three different contexts. Firstly, in the PAC model, an Occam algorithm is a learning algorithm which outputs a concept that is consistent with the examples as well as relatively simple compared to those examples. An efficient Occam algorithm can be shown to be an efficient PAC algorithm. Hence, if a learning algorithm can sufficiently *compress* the examples, it will automatically learn PAC.

Secondly, the Minimum Description Length principle tells us to select the simplest hypothesis, i.e., a hypothesis which most compresses the data. Under certain mild assumptions, this hypothesis selection scheme can be shown to be approximately equivalent to the optimal but infeasible hypothesis selection that is based on Bayes's Theorem. Accordingly, under those assumptions, Occam's Razor is an approximately optimal rule for hypothesis selection.

Thirdly and finally, Kolmogorov complexity was originally introduced in order to give a universal method for prediction. We are given an initial finite binary sequence, drawn according to some unknown computable "probability distribution", and we are to predict how the sequence will continue. We can predict this by combining the predictions of *all* computable distributions, weighing those predictions according to the Kolmogorov complexity of the distributions. Here distributions with low complexity are given high weight, in accordance with the Occamite preference for simplicity. This Occam-based prediction can be shown to converge very quickly to the true values.

5.5 Conclusion

The main aim of this thesis has been to examine the philosophical relevance of recent results from the field of computational learning theory. The models of learning put forward in that field are abstract and mathematical, but still capture much of what is important in "real world" learning. Accordingly, they can be used as (1) models of learning by human beings, and (2) as models of inductive theory construction in the empirical sciences. For the former case, the main application we discussed was a computational analysis of human language learning; for the latter, we discussed various formal justifications of Occam's Razor, using Kolmogorov complexity as a measure of simplicity. We feel that these results—as well as others that we discussed, and others we did *not* discuss—are highly relevant for philosophy and merit more attention than they presently receive. Let me end by expressing the hope that this thesis will contribute something to an increased awareness of formal learning theory among philosophers.

List of Symbols

$\in$	element
$\subseteq$	subset
$\subset$	proper subset
$\supseteq$	superset
$\supset$	proper superset
$\cup$	union
$\cap$	intersection
$\setminus$	set difference
Δ	symmetric difference
$\{x \mid C(x)\}$	set of all x that satisfy condition C
$\emptyset$	empty set
$\|S\|$	cardinality (number of elements) of set S
$\|r\|$	absolute value of number r
2^S	power set (set of all subsets) of set S
$S \times T$	Cartesian product of sets S and T
S^2	Cartesian product of set S with itself
$f : S \to T$	function f, with set S as domain and set T as range
$\mathbf{N}$	set of natural numbers
$\mathbf{Q}$	set of rational numbers
$\mathbf{R}$	set of real numbers
∞	infinity
$\sum_S$	summation over all members of set S
log	logarithm with base 2
ln	natural logarithm (base $e = 2.71\ldots$)
ε	empty string
S^*	set of all finite strings over alphabet S
$\{0,1\}^*$	set of all finite binary strings
S^∞	set of all infinite strings over alphabet S
$\{0,1\}^\infty$	set of all infinite binary strings
$\omega_{1:n}$	first n bits of infinite binary string ω
$X^{[n]}$	set of all strings of length at most n in domain X
$f^{[n]}$	projection of concept f on $X^{[n]}$
$\mathcal{F}^{[n]}$	projection of concept class $\mathcal{F}$ on $X^{[n]}$

$\mathbf{P}(A)$	probability of event A
$\mathbf{P}(A\|B)$	probability of A, given B
δ	confidence parameter
ε	error parameter
η	rate of malicious or random classification noise
$l_{min}(f, R)$	size (shortest name) of concept f, in representation R
$l_{min}(S, R)$	size of smallest concept consistent with examples S, in R
$\mathbf{D}_{VC}$	Vapnik-Chervonenkis dimension
$l(x)$	length of binary string x
$l(T)$	length of shortest program for Turing machine T
$K(x)$	Kolmogorov complexity of binary string x
$K(x\|y)$	Kolmogorov complexity of string x, given string y
Ω	halting probability
$\mathbf{m}$	universal distribution
$\mathbf{M}$	universal semimeasure

Bibliography

[Aar95] E. Aarts. *Investigations in Logic, Language and Computation.* PhD thesis, University of Utrecht, 1995.

[AB92] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, Cambridge, UK, 1992.

[AL88] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[All92] D. Allport. The changing relationship between AI programming languages and natural language processing formalisms. In R. Spencer-Smith and S. Torrance, editors, *Machinations: Computational Studies of Logic, Language, and Cognition*, pages 91–125. Ablex Publishing Corporation, Norwood, NJ, 1992.

[Ang87a] D. Angluin. Learning k-bounded context-free grammars. Technical Report YALEU/DCS/RR–557, Department of Computer Science, Yale University, 1987.

[Ang87b] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Ari60] Aristotle. *Posterior Analytics.* Harvard University Press, Cambridge, MA, 1960. Edited and translated by Hugh Tredennick.

[Ari76] R. Ariew. *Ockham's Razor: A Historical and Philosophical Analysis of Ockham's Principle of Parsimony.* PhD thesis, University of Illinois, Urbana-Champaign, 1976.

[Bac88] J. Bacon. Four modal modellings. *Journal of Philosophical Logic*, 17:207–220, 1988.

[Bac94] F. Bacon. *Novum Organum.* Open Court, Chicago, IL, 1994. Edited and translated by P. Urbach and J. Gibson. First published in 1620.

[BC74] H. Brandt Corstius. *Algebraïsche Taalkunde.* Oosthoek, Utrecht, 1974. In Dutch.

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[BGT91] R. Boyd, P. Gasper, and J. Trout, editors. *The Philosophy of Science*. MIT Press, Cambridge, MA, 1991.

[BJ89] G. S. Boolos and R. C. Jeffrey. *Computability and Logic*. Cambridge University Press, Cambridge, UK, third edition, 1989.

[Boa59] G. Boas. Some assumptions of Aristotle. *Transactions of the American Philosophical Society*, N. S. 49:5–98, 1959.

[Bot89] R. P. Botha. *Challenging Chomsky*. Basil Blackwell, 1989.

[Bun62] M. Bunge. The complexity of simplicity. *Journal of Philosophy*, 59:113–135, 1962.

[Bur80] E. A. Burtt. *The Metaphysical Foundations of Modern Science*. Routledge, London, revised edition, 1980. First edition 1924.

[Cam90] K. Campbell. *Abstract Particulars*. Basil Blackwell, Oxford, 1990.

[Car50] R. Carnap. *Logical Foundations of Probability*. Routledge & Kegan Paul, London, 1950.

[Car52] R. Carnap. *The Continuum of Inductive Methods*. The University of Chicago Press, Chicago, IL, 1952.

[Cha66] G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

[Cha69] G. J. Chaitin. On the length of programs for computing finite binary sequences: Statistical considerations. *Journal of the ACM*, 16:407–422, 1969.

[Cha74] G. J. Chaitin. Information-theoretic limitations of formal systems. *Journal of the ACM*, 21:403–424, 1974.

[Cha75] G. J. Chaitin. Randomness and mathematical proof. *Scientific American*, 232:47–52, May 1975.

[Cha87] G. J. Chaitin. *Algorithmic Information Theory*. Cambridge University Press, Cambridge, UK, 1987.

[Che86] C. Cherniak. *Minimal Rationality*. MIT Press, Cambridge, MA, 1986.

[Cho57] N. Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

[Cho65] N. Chomsky. *Aspects of the Theory of Syntax.* MIT Press, Cambridge, MA, 1965.

[Cho75] N. Chomsky. Quine's empirical assumptions. In Davidson and Hintikka [DH75], pages 53–68.

[Cho81] N. Chomsky. *Lectures on Government and Binding.* Fortis, Dordrecht, 1981.

[Cho83] N. Chomsky. On cognitive structures and their development: A reply to Piaget. In Piattelli-Palmarini [PP83].

[Cho86] N. Chomsky. *Knowledge of Language: Its Nature, Origin, and Use.* Praeger, 1986.

[Cho91] N. Chomsky. Linguistics and cognitive science: Problems and mysteries. In Kasher [Kas91], pages 26–53.

[Coh85] I. B. Cohen. *Revolution in Science.* Belknap/Harvard University Press, Cambridge, MA, and London, 1985.

[CT91] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley, New York, 1991.

[Den91] D. C. Dennett. Real patterns. *Journal of Philosophy*, 88:27–51, 1991.

[Der93] W. Derkse. *On Simplicity and Elegance: An Essay in Intellectual History.* PhD thesis, University of Amsterdam, 1993.

[DH75] D. Davidson and J. Hintikka, editors. *Words and Objections: Essays on the Work of W. V. Quine.* Reidel, revised edition, 1975.

[Dir63] P. A. M. Dirac. The evolution of the physicist's picture of the world. *Scientific American*, 208(5):45–53, 1963.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[GKPS85] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford, 1985.

[Göd31] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

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

[Goo72a] N. Goodman. *Problems and Projects.* Bobbs-Merrill, Indianapolis and New York, 1972.

[Goo72b] N. Goodman. Safety, strength, simplicity. In *Problems and Projects* [Goo72a], pages 334–336. First published in 1961.

[Goo72c] N. Goodman. The test of simplicity. In *Problems and Projects* [Goo72a], pages 279–294. First published in 1958.

[Goo72d] N. Goodman. Uniformity and simplicity. In *Problems and Projects* [Goo72a], pages 347–354. First published in 1967.

[Goo83] N. Goodman. *Fact, Fiction, and Forecast.* Harvard University Press, Cambridge, MA, fourth edition, 1983.

[Har93] R. A. Harris. *The Linguistics Wars.* Oxford University Press, New York, 1993.

[Haw88] J. A. Hawkins, editor. *Explaining Language Universals.* Basil Blackwell, Oxford, 1988.

[Hem45a] C. G. Hempel. Studies in the logic of confirmation (part I). *Mind*, 54(213):1–26, 1945.

[Hem45b] C. G. Hempel. Studies in the logic of confirmation (part II). *Mind*, 54(214):97–121, 1945.

[Hem66] C. G. Hempel. *Philosophy of Natural Science.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[Hes67] M. B. Hesse. Simplicity. In P. Edwards, editor, *The Encyclopedia of Philosophy*, pages 445–448, Volume 7. Macmillan, London, 1967.

[Hof79] D. R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books, New York, 1979.

[Hom24] Homer. *The Iliad.* Harvard University Press, Cambridge, MA, 1924. Translated by A. T. Murray. Two volumes.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, MA, 1979.

[Hum56] D. Hume. *An Enquiry Concerning Human Understanding.* Gateway edition, Chicago, IL, 1956. First published in 1748.

[Hum61] D. Hume. *A Treatise of Human Nature.* Dolphin Books. Doubleday, 1961. First published in 1739–1740.

[Ish90] H. Ishizaka. Polynomial time learnability of simple deterministic languages. *Machine Learning*, 5(2):151–164, 1990.

[Jev74] W. S. Jevons. *The Principles of Science: A Treatise.* Macmillan, London, 1874.

[Kas91] A. Kasher, editor. *The Chomskyan Turn.* Basil Blackwell, 1991.

[Kol65] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7, 1965.

[Kol68] A. N. Kolmogorov. Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, IT-14(5):662–664, 1968.

[Kuh77] T. Kuhn. Second thoughts on paradigms. In F. Suppe, editor, *The Structure of Scientific Theories*, pages 459–482. University of Illinois Press, second edition, 1977.

[KV94] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1994.

[Lai88] P. D. Laird. *Learning from Good and Bad Data.* Kluwer Academic Publishers, Boston, MA, 1988.

[Lak71] I. Lakatos. History of science and its rational reconstructions. In R. C. Buck and R. S. Cohen, editors, *Philosophy of Science Association 1970*, volume 8 of *Boston Studies in the Philosophy of Science*, pages 91–136. Reidel, Dordrecht, 1971.

[Lam89] M. van Lambalgen. Algorithmic information theory. *Journal of Symbolic Logic*, 54(4):1389–1400, 1989.

[Loc93] J. Locke. *An Essay Concerning Human Understanding.* Everyman Library, David Campbell Publishers, London, abridged edition, 1993. First published in 1690.

[Lok91] G. J. C. Lokhorst. Is de mens een eindige automaat? In F. Geraedts and L. de Jong, editors, *Ergo Cogito III: Portret van een generatie*, pages 89–105. Historische Uitgeverij Groningen, 1991.

[LV97] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, Berlin, second edition, 1997.

[Lyo95] W. Lyons, editor. *Modern Philosophy of Mind.* Everyman, London, 1995.

[McA96] J. W. McAllister. *Beauty & Revolution in Science.* Cornell University Press, Ithaca, NY, and London, 1996.

[Mil58] J. S. Mill. *A System of Logic, Ratiocinative and Inductive.* Harper, New York, 1858.

[Min68] M. L. Minsky, editor. *Semantic Information Processing.* MIT Press, Cambridge, MA, 1968.

[ML66] P. Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

[Nat91] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, San Mateo, CA, 1991.

[New91] F. J. Newmeyer. Rules and principles in the historical development of generative syntax. In Kasher [Kas91], pages 200–230.

[NW97] S-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin, May 1997.

[Pai82] A. Pais. *'Subtle is the Lord...' The Science and the Life of Albert Einstein.* Oxford University Press, Oxford, 1982.

[Pas67] J. A. Passmore. *A Hundred Years of Philosophy.* Penguin, second edition, 1967.

[Pei58] C. S. Peirce. *Collected Papers.* Harvard University Press, Cambridge, MA, 1958. Edited by C. Harstshorne and P. Weiss. Volumes I–VII.

[Pen89] R. Penrose. *The Emperor's New Mind, Concerning Computers, Minds, and the Laws of Physics.* Oxford University Press, Oxford, 1989.

[Pin84] S. Pinker. *Language Learnability and Language Development.* Harvard University Press, Cambridge, MA, 1984. The 1996 edition contains new commentary by Pinker.

[Pin94] S. Pinker. *The Language Instinct.* Penguin Books, 1994.

[Pla95] U. T. Place. Is consciousness a brain process? In Lyons [Lyo95], pages 106–116. First published in 1956.

[Pop59] K. R. Popper. *The Logic of Scientific Discovery.* Hutchinson, London, 1959.

[Pos64] P. M. Postal. Limitations of phrase structure grammars. In J. A. Fodor and J. J. Katz, editors, *The Structure of Language: Readings in the Philosophy of Language*, pages 137–151. Prentice Hall, 1964.

[PP83] M. Piattelli-Palmarini, editor. *Language and Learning: The Debate Between Jean Piaget and Noam Chomsky.* Routledge, London, 1983.

[PR73] S. Peters and R. Ritchie. On the generative power of transformational grammars. *Information Sciences*, 6:49–83, 1973.

[PS97] A. Prince and P. Smolensky. Optimality: From neural networks to universal grammar. *Science*, 275:1604–1610, 14 March 1997.

[Put71] H. Putnam. The "innateness hypothesis" and explanatory models in linguistics. In J. Searle, editor, *The Philosophy of Language.* Oxford University Press, New York, 1971.

[Put83] H. Putnam. What is innate and why: Comments on the debate. In Piattelli-Palmarini [PP83], pages 287–309.

[Qui60] W. V. O. Quine. *Word and Object*. MIT Press, Cambridge, MA, 1960.

[Qui69] W. V. O. Quine. Natural kinds. In *Ontological Relativity and Other Essays*, pages 114–138. Columbia University Press, New York, 1969. Reprinted in [BGT91], pages 159–170.

[Qui75] W. V. O. Quine. Reply to Chomsky. In Davidson and Hintikka [DH75], pages 302–311.

[Qui76] W. V. O. Quine. On simple theories of a complex world. In *The Ways of Paradox and Other Essays*, pages 255–258. Harvard University Press, Cambridge, MA, revised and enlarged edition, 1976.

[Rei49] H. Reichenbach. *The Theory of Probability*. University of California Press, Berkeley, 1949.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Ris89] J. J. Rissanen. *Stochastical Complexity and Statistical Inquiry*, volume 15 of *Series in Computer Science*. World Scientific, Singapore, 1989.

[Ros78] H. Rosemont. Gathering evidence for linguistic innateness. *Synthese*, 38:127–148, 1978.

[Rus48] B. Russell. *Human Knowledge: It's Scope and Limits*. George Allen and Unwin, London, 1948.

[Rus80] B. Russell. *The Problems of Philosophy*. Oxford University Press, Oxford, 1980. First published in 1912.

[Rus91] S. Russell. Inductive learning by machines. *Philosophical Studies*, 64:37–64, 1991.

[Sei97] M. S. Seidenberg. Language acquisition and use: Learning and applying probabilistic constraints. *Science*, 275:1599–1603, 14 March 1997.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technology Journal*, 27:379–423, 623–656, 1948.

[Slo95] R. H. Sloan. Four types of noise in PAC learning. *Information Processing Letters*, 54:157–162, 1995.

[Sma95] J. J. C. Smart. Sensations and brain processes. In Lyons [Lyo95], pages 117–132. First published in 1959.

[Sob75] E. Sober. *Simplicity*. Clarendon Press, Oxford, 1975.

[Sol64] R. J. Solomonoff. A formal theory of inductive inference, part 1 and 2. *Information and Control*, 7:1–22, 224–254, 1964.

[Tha90] P. Thagard. Philosophy and machine learning. *Canadian Journal of Philosophy*, 20(2):261–276, 1990.

[Tho18] W. M. Thornburn. The myth of Ockham's razor. *Mind*, 27:345–353, 1918.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungproblem. In *Proceedings of the London Mathematical Society*, volume 42, pages 230–265, 1936. Correction, ibidem (vol. 43), pages 544–546.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Val85] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 560–566. Morgan Kaufmann, 1985.

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[Wei64] J. Weinberg. *A Short History of Medieval Philosophy*. Princeton University Press, Princeton, NJ, 1964.

[Wil66] D. C. Williams. *Principles of Empirical Realism*. Charles C. Thomas, Springfield, 1966.

[Wit53] L. Wittgenstein. *Philosophische Untersuchungen*. Suhrkamp, 1953. Edition 1995, published together with Tractatus and Tagebücher.

Index of Names

Index of Subjects