Logical Prior Probability

Abram Demski*

Institute for Creative Technologies, 12015 Waterfront Drive, Playa Vista, CA 90094

Abstract. A Bayesian prior over first-order theories is defined. It is shown that the prior can be approximated, and the relationship to previously studied priors is examined.

1 Introduction & Motivation

The purpose of this paper is to present a prior over theories in first-order logic, similar in nature to the priors of algorithmic probability. There are several possible motivations for such a prior. First, it is hoped that the study of priors over logics will be useful to the study of realistic reasoning. Probabilistic reasoning over logic gives us a structure of inference which is not as evident in non-logical universal priors. Second, logical theories may be easier to examine than other possible knowledge representations, motivating the learning of logical theories as a goal in itself (independent of prediction accuracy and other concerns). In this case, a theory of universal learning via logical theories may be useful. Third, the logical prior presented here may give some benefits even if the only consideration is prediction accuracy.

The primary idea is that of the random theory. By building up first-order theories one random sentence at a time, a probability that a particular sentence becomes true can be defined.

One way of motivating the approach is to consider what would happen if we attempted to apply the universal semidistribution $\mathcal{M}$ to beliefs in predicate calculus. (I will rely on some concepts which are explained more fully in section 2.) $\mathcal{M}$ is a prior over bit-sequences. We can encode our beliefs about propositions as beliefs about sequences, by giving each sentence a number n (as is done in Gödel numbering, [1]), and using the bit at position n to represent the truth or falsehood of that sequence. Suppose we have an observation set, Σ, of sentences which we've accepted as true. We would like to know how to assign probability to the other sentences. The obvious approach is to update $\mathcal{M}$ on the bits representing Σ. Two main problems arise:

- Consistency & Completeness. $\mathcal{M}$ does not know that the bits represent logical sentences, so it will not assign probability based on the logical consequences of Σ. For example, for each $A \in \Sigma$, some probability will still be

* This effort has been sponsored by the U.S. Army and the Air Force Office of Scientific Research. Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

assigned to the negation of A. We would like to assign probability 1 to the consequences of Σ, and probability 0 to things inconsistent with Σ.
- Non-sequential enumeration. $\mathcal{M}$ is a mixture distribution composed of programs which output the bits of the sequentially. Σ will have recursively enumerable consequences, but due to the undecidability of the consequence relation, it will not be possible in general to enumerate these consequences in linear order.

The second problem is more subtle than the first, but follows from it: if a distribution got the logical consequences right, then it would be enumerating them properly. The point is seperated out because it is an interesting divergence form $\mathcal{M}$. To illustrate this issue, suppose that we want to define $\mathcal{M}'$ which is a mixture distribution over arbitrary computable enumerations of bits, rather than only sequential enumerations. We understand the programs as printing a sequence of (location, bit) pairs, and take each pair to set the bit of the sequence at the given location.

To make $\mathcal{M}'$ well-defined, we need to decide what to do when conflicting pairs are given by a program. A program may print the pair (40,1) and later print (40,0). What contribution should the program make to the probability of that bit?

Three options are:

$\mathcal{M}'_1$: The earliest pair for a given location is used.

$\mathcal{M}'_2$: The program is thrown out when it produces conflicting pairs. It no longer contributes anything to the distribution.

$\mathcal{M}'_3$: The latest pair for a location is used. If the program keeps printing conflicting bits for a location forever, it is not considered to contribute any probability for the distribution of that location (just as if it had never printed any pair for that location).

The resulting priors are arranged in order of expressive power. $\mathcal{M}'_2$ contains any model which $\mathcal{M}'_1$ does, since we can wrap an $\mathcal{M}'_1$ program in an output-checker which keeps the program from printing any pair for a previously-set location. $\mathcal{M}'_3$ subsumes $\mathcal{M}'_2$, since we can replicate the behavior of "throwing out" a program by printing conflicting pairs for all locations forever. Also, $\mathcal{M}'_1$ subsumes $\mathcal{M}$, since we can deal with locations in sequential order.

Thus, we can establish $\mathcal{M} \leq \mathcal{M}'_1 \leq \mathcal{M}'_2 \leq \mathcal{M}'_3$ (where $\leq$ indicates multiplicative dominance, to be defined) without too much trouble. It seems reasonable to further conjecture $\mathcal{M} < \mathcal{M}'_1 < \mathcal{M}'_2 < \mathcal{M}'_3$.

$\mathcal{M}'_3$ is related to generalized Kologorov complexity as discussed in [6], which shows that such a distribution cannot be approximated. As such, it is not clear how useful it might be to the study of intelligence.

Since consistency & completeness have not yet been dealt with, these distributions are better thought of as alternative sequence prediction priors, rather than trying to interpret them as distributions over logical theories by the previously-mentioned numbering.

Enforcing both consistency and completeness will result in logical priors which look similar to the one to be described: a process generating random sentences is constrained in such a way as to guarantee that the results make sense in terms of the logic.

2 Selected Background and Notation

2.1 First-Order Logic

We will be using first-order logic, defining the language $\mathcal{L}$ of first-order sentences as follows:

- There is an infinite stock of variable symbols, $v_1, v_2, ... \in \mathcal{V}$, an infinite stock of predicate symbols, $p_1, p_2, ... \in \mathcal{P}$, and an infinite stock of function symbols, $f_1, f_2, ... \in \mathcal{F}$.
- The number of arguments fed to a predicate or function is referred to as its *arity*. For example, a predicate of arity 2 is typically referred to as a relation. A function of arity 0 is referred to as a constant, and a predicate of arity 0 is a proposition. For simplicity, the arity of a symbol will be inferred from its use here, rather than set ahead of time. If the same symbol is used with multiple arities, the uses are independent (so f_2 would notate distinct functions in $f_2(v_1)$ versus $f_2(v_1, v_2)$).
- An *expression* is a composition of function symbols and variable symbols, for example $f_1(f_1(v_1))$. Specifically, the set of expressions $\mathcal{E}$ are defined inductively by: $\mathcal{V} \subset \mathcal{E}$, and for every function $f_n \in \mathcal{F}$ of arity a and expressions $e_1, e_2, ..., e_a \in \mathcal{E}$, we have $f_n(e_1, e_2, ...e_a) \in \mathcal{E}$.
- For $e_1, e_2 \in \mathcal{E}$, $e_1 = e_2$ is in $\mathcal{L}$; this represents equality.
- For $p_n \in \mathcal{P}$ of arity a and $e_1, e_2, ..., e_a \in \mathcal{E}$, we have $p_n(e_1, e_2, ..., e_a) \in \mathcal{L}$.
- For $A, B \in \mathcal{L}$, we have $(A \wedge B) \in \mathcal{L}$ and $(A \vee B) \in \mathcal{L}$; these represent conjunction and disjunction, respectively. (Parentheses will be omitted in this document when the intended grouping is clear.)
- For $S \in \mathcal{L}$, we have $\neg(S) \in \mathcal{L}$. This represents negation. (Again, parentheses may be omitted.)
- For any $S \in \mathcal{L}$ and $v_n \in \mathcal{V}$, we have $\forall v_n.(S) \in \mathcal{L}$ and $\exists v_n.(S) \in \mathcal{L}$, representing universal and existential quantification. (Parentheses may be ommited.)

If sentence A logically implies sentence B (meaning, B is true in any situation in which A is true), then we write $A \vDash B$. The notation also applies to multiple premises; if A and B together imply C, we can write $A, B \vDash C$. Uppercase greek letters will also be used to denote sets of sentences. We can write $A \nvDash B$ to say that A does not logically imply B.

If A implies B according to the inference rules (meaning, we can derive B starting with the assumption A), we write $A \vdash B$. This notation applies to multiple premises as well, and can be denied as $\nvdash$.

The inference rules will not be reviewed here, but some basic results will be important. These results can be found in many textbooks, but in particular, [1] has material on everything mentioned here.

Soundness. For a sentence S and a set of sentences Γ, if $\Gamma \vdash S$, then $\Gamma \vDash S$. That is, the inference rules will never derive something that doesn't logically follow from a set of premises.

Completeness. For a sentence S and a set of sentences Γ, if $\Gamma \vDash S$, then $\Gamma \vdash S$. That is, the inference rules can derive anything which logically follows from a set of premises.

Since the rules for $\vdash$ can be followed by a computer, this shows that $\vdash$ is computably enumerable: a (non-halting) program can enumerate all the true instances of $\Gamma \vdash S$.

Undecidability. For a given Γ and S, no general procedure exists which can decide whether $\Gamma \vdash S$ or $\Gamma \nvdash S$. Completeness implies that we can know $\Gamma \vdash S$ if it is true; however, if it is not, there is no general way to determine $\Gamma \nvdash S$.

Encoding computations. Any computable function can be encoded in first-order logic. This can be done, for example, by providing axioms related to the behavior of Turing machines.

2.2 Algorithmic Information Theory

$\mathcal{B}$ denotes the binary alphabet, $\{0, 1\}$; $\mathcal{B}^n$ denotes the set of binary strings of length n; $\mathcal{B}^*$ denotes the set of binary strings of any finite length; $\mathcal{B}^\infty$ denotes the set of binary strings of infinite length; and $S_\mathcal{B} = \mathcal{B}^* \cup \mathcal{B}^\infty$ denotes the set of finite and infinite binary strings. String concatenation will be represented by adjacency, so ab is the concatenation of a and b.

Consider a class C_1 of Turing machines with three or more tapes: an input tape, one or more work tapes, and an output tape. The input and output tape are both able to move in just one direction. Any Turing machine $T \in C_1$ defines a partial function f_T from $\mathcal{B}^\infty$ to $S_\mathcal{B}$: for input $i \in \mathcal{B}^\infty$, $f_T(i)$ is considered to be the string which T writes to the output tape, which may be infinite if T never stops writing output. Now consider a *universal* machine from this class; that is, a machine $U \in C_1$ such that for any other machine $T \in C_1$, there is a finite sequence of bits $s \in \mathcal{B}^*$ which we can place on U's input tape to get it to behave exactly like T; that is, $f_U(si) = f_T(i)$ for all i.

A distribution $\mathcal{M}$ over $S_\mathcal{B}$ can be defined by feeding random bits to U; that is, we take $f_U(i)$ for uniformly random $i \in \mathcal{B}^\infty$.[1]

The development here has been adapted from [5].

Now, how do we compare two distributions?

P_1 *multiplicatively dominates* P_2 iff there exists $\alpha > 0$ such that $P_1(x) > \alpha P_2(x)$ for any x. An intuitive way of understanding this is that P_1 needs at most

[1] $\mathcal{M}$ is not actually a probability distribution, but rather, a semimeasure. The Solomonoff distribution is a probability distribution defined from $\mathcal{M}$: we apply the *Solomonoff normalization* to $\mathcal{M}$, which gives a distribution over $\mathcal{B}^\infty$. The details of normalization will not be given here.

a constant amount more evidence to reach the same conclusion as P_2.[2] *Strict multiplicative dominance* means that P_1 multiplicatively dominates P_2, but the reverse is not the case. This indicates that P_1 needs at most a constant amount more evidence to reach the same conclusion as P_2, but we can find examples where P_2 needs arbitrarily more evidence than P_1 to come to the conclusion P_1 reaches.

The main reason $\mathcal{M}$ is interesting is that it is multiplicatively dominant over any computable probability distribution for sequence prediction. This makes it a highly general tool.

P_1 *exponentially dominates* P_2 iff there exists $\alpha, \beta > 0$ such that $P_1(x) > \alpha P_2(x)^\beta$. This intuitively means that P_1 needs at most some constant multiple of the amount of evidence which P_2 needs to reach a specific conclusion. *Strict* exponential dominance again indicates that the reverse is not the case, which means that P_2 needs *more* than multiplicatively more evidence to reach some conclusions that P_1 can reach.

We can also define (multiplicative or exponential) *equivalence*: two distributions are considered equivalent when they mutually dominate each other.

3 A Notion of Logical Probabilities

3.1 Requirements

I will follow [7] in the development of the idea of a probability distribution over a language, since this provides a particularly clear idea of what it means for a continuous-valued belief function to fit with a logic. I shall say that the distribution *respects the logic*. The approach is to define probability as a function on sentences in a language, rather than by the more common σ-algebra approach, and require the probabilities to follow several constraints based on the logic. Since we are using classical logic, I will simplify their constraints for that case.

Let $\mathcal{L}$ be the language of first-order logic from section 2. We want a probability function $P : \mathcal{L} \to \mathbb{R}$ to obey the following rules:

(P0) $P(A) = 0$ if A is refutable.
(P1) $P(A) = 1$ if A is provable.
(P2) If A logically implies B, then $P(A) \leq P(B)$.
(P3) $P(A) + P(B) = P(A \vee B) + P(A \wedge B)$.

From these, we can prove other typical properties such as $P(A) + P(\neg A) = 1$.

3.2 Definition As a Generative Process

The idea behind the prior is to consider theories as being generated by choosing sentences at random, one after another. The probability of a particular sentence

[2] This is true if we measure evidence by the log of the likelihood ratio. $P_1(x|e) = P_1(x)P_1(e|x)/P_1(e)$, so multiplicative dominance indicates that$P_1(e|x)/P_1(e)$ doesn't have to get too extreme to bridge the distance between P_1 and P_2.

is taken to be the probability that it occurs in a theory randomly generated in this manner.

To be more precise, suppose we have some random process to generate individual sentences from our language $\mathcal{L}$. This generation process will be denoted $\mathcal{G}$, and the probability that S_1, S_2, ..., S_n are the first n statements generated will be written $\mathcal{G}(S_1, S_2, ..., S_n)$. $\mathcal{G}$ could be a highly structured process such as the $\mathcal{M}'$ distributions mentioned in section 1, but this seems unecessarily complicated.[3] Unless otherwise mentioned, this paper will define $\mathcal{G}$ based on a simple probabilistic grammar on sentences, which generates sentences recursively by selecting each syntactic element given in section 2.1 with some probability. When selecting from the variable, predicate, or function symbols, the subscript number must be constructed, for example by assigning $\frac{1}{11}$ chance to each digit and $\frac{1}{11}$ chance to terminating the digit string. We define $\mathcal{G}(S_1, S_2, ..., S_n) = \Pi_{i=1}^{n} \mathcal{G}(S_i)$.

A theory is a set of sentences in $\mathcal{L}$. To generate a random theory, we generate a sequence of sentences S_1, S_2, S_3, ... according to the following process. For each S_n, use sentences from $\mathcal{G}$, but discarding those which are inconsistent with the sentences so far; that is, rejecting any candidate for S_n which would make $S_1 \wedge ... \wedge S_n$ into a contradiction. (For S_1, the set of preceding sentences is empty, so we only need to ensure that it does not contradict itself.)

Notice that there is no stopping condition. The sequence generated will be infinite. However, the truth or falsehood of any particular statement (or any finite theory) will be determined after a finite amount of time. (The remaining sentences generated will either be consequences of, or irrelevant to, the statement in question.) Shorter (finite) theories will have a larger probability of occurring in the sequence.

In this way, we induce a new probability distribution P_L on sentences from the one we began with, $\mathcal{G}$. $P_L(S)$ is the probability that a sentence S will be present in a sequence S_1, S_2, S_3, ... generated from $\mathcal{G}$ as described. Unlike $\mathcal{G}$, P_L respects the logic:

Theorem 1. *P_L obeys (P0)-(P3).*

Proof. (P0) is satisfied easily, since the process explicitly forbids generation of contradictions. (P1) is satisfied, because a provable statement can never contradict the sentences so far, so each will eventually be generated by chance as we continue to generate the sequence. Therefore, provable statements are generated with probability 1. (P2) is satisfied, by a similar argument: if we have already generated A, but A implies B, then anything which contradicts B will contradict A, and hence never be generated. This means that B will never be ruled out, and so must eventually be generated at random.[4] Therefore the probability for B is at least as high as that if A.

[3] If we did choose to use these, we would need to address the fact that they are only semimeasures, not full probability distributions.

[4] Notice, this means any theory generated in this manner will contain all of its logical consequences with probability 1. This allows us to talk just about what sentences are in the theory, when we might otherwise need to talk about the theory plus all its logical consequences.

We can extend the argument a bit further to show (P3).

Since $A \vdash A \vee B$ and $B \vdash A \vee B$, the sentence $A \vee B$ will occur in any theory in which A or B occurs. Moreover, if $A \vee B$ occurs, then it would be inconsistent for both $\neg A$ and $\neg B$ to occur later. As a result, either A or B will eventually occur. So $P_L(A \vee B)$ equals the probability that either A or B occurs.

If both A and B occur in a theory, then $\neg(A \wedge B)$ would be contradictory, so will not occur; therefore, $A \wedge B$ will eventually be generated. On the other hand, if $A \wedge B$ occurs in a sequence, it would be inconsistent for either $\neg A$ or $\neg B$ to occur, so both A and B will eventually be present. $P_L(A \wedge B)$ equals the probability that both A and B occur in a sequence.

Since $P_L(A \vee B)$ equals the probability that *either* A or B occurs, and $P_L(A \wedge B)$ equals the probability that *both* A and B occur, we have $P_L(A \vee B) = P_L(A) + P_L(B) - P_L(A \vee B)$. This proves (P3). □

The conditional probability can be defined as usual, with $P_L(A|B) = P_L(A \wedge B)/P_L(B)$. We can also extend the definition of $P_L()$ to include probabilities of sets of sentences, so that $P_L(\Gamma)$ for $\Gamma \subset \mathcal{L}$ is the probability that all $S \in \Gamma$ will be present in a sequence generated by the process defined above. (By an argument similar to the one used to prove (P3), the probability of a set of sentences will be equal to the probability of the conjunction.)

3.3 Approximability

The generative process described so far cannot be directly implemented, since there is no way to know for sure that a theory remains consistent as we add sentences at random. However, we can asymptotically approach $P_L()$ by eliminating inconsistent possibilities when we find them.

I assume in this section that $\mathcal{G}$ is such that we can sample from it. It may be possible that some interesting choices of $\mathcal{G}$ result in an approximable P_L without a sampleable $\mathcal{G}$.

Suppose we want to approximate $P_L(A)$. I shall call a partial sequence S_1, S_2, ..., S_n a *prefix*. Consider the following Monte Carlo approximation:

```
t=1, y=1, n=1.
loop:
    // Reset the prefix at the beginning of each loop.
    prefix=none
    // Until we get A or neg(A),
    while not (s=A or s=neg(A)):
        // Get a random sentence.
        s=generate()
        // Append sample to the sequence so far.
        prefix=push(s,prefix)
        // Spend time t looking for contradictions.
        c=check(prefix,t)
        // If a contradiction is found,
```

```
    if c:
        // backtrack.
        pop(prefix)
// If the generated prefix contains A,
if (s=A):
    // increment y.
    y=y+1
// Otherwise,
else:
    // increment n.
    n=n+1
// Increment t at the end of each loop.
t=t+1
```

The variables y and n count the number of positive and negative samples, while t provides a continually rising standard for consistency-detection on the sampled prefixes. (I will use the term "sample" to refer to generated prefixes, rather than individual sentences.) To that end, the function check(,) takes a prefix and an amount of time, and spends that long trying to prove a contradiction from the prefix. If one is found, check returns true; otherwise, false. The specific proof-search technique is of little consequence here, but it is necessary that it is exhaustive (it will eventually find a proof if one exists). The function neg() takes the negation; so, we are waiting for either A or $\neg A$ to occur in each sample. The prefix is represented as a FILO queue. push() adds a sentence to the given prefix, and pop() removes the most recently added sentence.

The inner loop produces individual extensions at random, backtracking whenever an inconsistency is found. The loop terminates when a theory includes either A or $\neg A$. The outer loop then increments y or n based on the result, increments t, erases the prefix, and re-enters the inner loop to get another sample.

Theorem 2. *$\frac{y}{n+y}$ will approach $P_L(A)$.*

Proof. Since every inconsistency has a finite amount of time required for detection, the probability of an undetected inconsistency will fall arbitrarily far as t rises. The probability of consistent samples, however, does not fall. Therefore, the counts will eventually be dominated by consistent samples.

The question reduces to whether the probability of a consistent sample containing A is equal to $P_L(A)$. We can see that this is the case, since if we assume that the generated sentences will be consistent with the sentence so far, then the generation probabilities are exactly those of the previous section. □

3.4 Comparison

It would be interesting to know how this prior compares with the priors which have been defined via Turing machines.

In order to compare the first-order prior with priors for sequence prediction, we need to apply the first-order prior to sequence prediction. We can do so

by encoding bit sequences in first-order logic. For example, f_1 can serve as a logical constant representing the sequence to be observed and predicted; $f_2()$ can represent adding a 0 to the beginning of some sequence; and $f_3()$ can represent adding a 1. So, to say that the sequence begins "0011..." we would write $f_1 = f_2(f_2(f_3(f_3(f_4))))$, where f_4 is a logical constant standing for the remainder of the sequence. The probability of a bit sequence can be taken as the probability of a statement asserting that bit sequence. Define P_{LS} to be the resulting prior over bit sequences.

It seems possible to show that P_{LS} is exponentially equivalent to $\mathcal{M}'_2$ from section 1. $\mathcal{M}'_2$ will dominate P_{LS}, because P_{LS} can be defined by a Turing machine which takes an infintie stream of random bits, interpretes them as first-order sentences, and outputs all (location, bit) pairs which follow deductively from them. Since $\mathcal{M}'_2$ is constructed from a universal Turing machine, it will have this behavior with some probability. Inconsistent theories will start outputting inconsistent pairs, and so will not be included in $\mathcal{M}'_2$. Thus we get the behavior of P_{LS}. On the other hand, since first-order logic can encode computations, it seems that we can encode all the enumerations included in $\mathcal{M}'_2$. However, the encoding may not be efficient enough to get us multiplicative dominance. Exponential dominance seems possible to establish, since the expression-length of the representation of a bit-tape in first-order logic will be linear in the bit-length of that tape.

Since this development is insufficiently formal, the statement remains a conjecture here.

4 Conclusion & Questions

One hopeful application of this prior is to human-like mathematical reasoning, formalizing the way that humans are able to reason about mathematical conjectures. The study of conjecturing in artificial intelligence has been quite successful[5], but it is difficult to analyse this theoretically, especially from a Bayesian perspective.

This situation springs from the problem of *logical omniscience* [3]. The logical omniscience problem has to do with the sort of uncertainty that we can have when we are not sure what beliefs follow from our current beliefs. For example, we might understand that the motion of an object follows some particular equation, but be unable to calculate the exact result without pen and paper. Because the brain has limited computational power, we must expect the object to follow some plausible range of motion based on estimation. Standard probability theory does not model uncertainty of this kind. A distribution which follows the laws of probability theory will already contain all the consequences of any beliefs (it is logically omniscient). Real implementations cannot work like that.

> An agent might even have beliefs that logically contradict each other. Mersenne believed that $2^{67}-1$ is a prime number, which was proved false

[5] For example, AM[4] and Graffiti[2].

> in 1903, [...] Together with Mersenne's other beliefs about multiplication and primality, that belief logically implies that $0 = 1$. [3]

Gaifman proposes a system in which probabilities are defined only with respect to a finite subset of the statements in a language, and beliefs are required to be consistent only with respect to chains of deduction in which each statement occurs in this limited set.

I will not attempt to address this problem to its fullest here, but approximations to P_L such as the one in section 3.3 seem to have some good properties in this area. If the proof of some statement X is too long for some approximation $A(t, X, Y)$ to $P_L(X|Y)$ to find given time t, then S will be treated exactly like a statement which is not provable: it will be evaluated with respect to how well it fits the evidence Y, given the connections which $A(t, X, Y)$ can find within time t. For example, if some universal statement $\forall x.S[x]$ can be proven from Y, but the proof is too long to find in reasonable time, then the probability of $\forall x.S[x]$ will still tend to rise with the number of individual instances $S[i]$ which are found to be true (although this cannot be made precise without more assumptions about the approximation process).

It is not clear how one would study this problem in the context of Solomonoff induction. Individual "beliefs" are not easy to isolate from a model when the model is presented as an algorithm. The problem of inconsistent beliefs does not even arise.

I do not claim that the first-order prior is a complete solution to this problem. For example, we do *not* get the desirable property that as we see arbitrarily many instances of a particular proposition, the probability of the universal generalization goes to 1. This fits with the semantics of first-order logic, but seems to be undesirable in other cases.

References

1. G. Boolos, J.P. Burgess, and R.C. Jeffrey. *Computability and Logic*. Cambridge University Press, 2007.
2. Dinneen M. Brewster, T. and V. Faber. A computational attack on the conjectures of graffiti: New counterexamples and proofs. *Discrete Mathematics*, 147:1–3, 1992.
3. H. Gaifman. Reasoning with limited resources and assigning probabilities to arithmetical statements. *Synthese*, 140(1951):97–119, 2004.
4. D. Lenat and J. Brown. Why am and eurisko appear to work. *Artificial Intelligence*, 23(3):269 – 294, 1984.
5. M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications (2. ed.)*. Graduate texts in computer science. Springer, 1997.
6. J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science*, 13(4):587–612, 2002.
7. B. Weatherson. From classical to intuitionistic probability. *Notre Dame Journal of Formal Logic*, 44(2):111–123, April 2003.