Incompleteness and Artificial Intelligence

Shane Legg[⋆]

Dalle Molle Institute for Artificial Intelligence
Galleria 2, Manno-Lugano 6928, Switzerland
shane@idsia.ch

1 Introduction

The implications of Gödel incompleteness for artificial intelligence have been studied many times before (e.g. [8]), as has the relation between Kolmogorov complexity and incompleteness (e.g. [3, 2]). In this paper we look at the relationship between these from a new angle. We start by considering the prediction of computable sequences of bounded complexity, an ability which would be fundamental to an artificial intelligence. We then show that although very powerful algorithms exist which can solve this problem, they are necessarily very complex. From this, we then prove that it is impossible to find any of these powerful algorithms due to Gödel incompleteness. This places serious limitations on our ability to understand and analyse intelligent systems using mathematics, and adds a new perspective on the relation between artificial intelligence, complexity and Gödel incompleteness.

2 Preliminaries

Let $\mathbb{B} := \{0, 1\}$, let $\mathbb{B}^*$ be the set of (finite) binary strings, and let $\mathbb{B}^\infty$ be the set of (infinite) binary sequences. A *substring* of x is defined $x_{j:k} := x_j x_{j+1} \ldots x_k$ where $1 \leq j \leq k \leq n$. By $|x|$ we mean the length of the string x, for example, $|x_{j:k}| = k-j+1$. We will sometimes need to encode a natural number as a string. Using simple encoding techniques it can be shown that there exists a computable injective function $f : \mathbb{N} \to \mathbb{B}^*$ such that $\forall n \in \mathbb{N} : |f(n)| \leq \log_2 n + 2\log_2 \log_2 n + 1$. We represent a monotone universal Turing machine by $\mathcal{U}$ [6].

Definition 1. *A sequence $\omega \in \mathbb{B}^\infty$ is a* **computable sequence** *if $\exists q \in \mathbb{B}^* : \mathcal{U}(q) = \omega$. We denote the set of all computable sequences by $\mathcal{C}$.*

Definition 2. *A* **computable predictor** *is an algorithm $p \in \mathbb{B}^*$ that on a universal Turing machine $\mathcal{U}$ computes a total function $\mathbb{B}^* \to \mathbb{B}$.*

We will often write $p(x)$ to mean the function computed by the program p when executed on $\mathcal{U}$ along with the input string x, that is, $p(x)$ is short hand for $\mathcal{U}(p, x)$. Having $x_{1:n}$ as input, the objective of a predictor is for its output, called

⋆ This work was funded by grant SNF 200020-107616.

its *prediction*, to match the next symbol in the sequence. Formally we express this by writing $p(x_{1:n}) = x_{n+1}$.

As the algorithmic prediction of incomputable sequences, such as the halting sequence, is impossible by definition, we only consider the problem of predicting computable sequences. We will assume that the predictor has an unlimited supply of computation time and storage.

Definition 3. *We say that a predictor p can* **learn to predict** *a sequence $\omega := x_1 x_2 \ldots \in \mathbb{B}^\infty$ if $\exists m \in \mathbb{N}, \forall n \geq m : p(x_{1:n}) = x_{n+1}$.*

The existence of m in the above definition need not be constructive, that is, we might not know when the predictor will stop making prediction errors for a given sequence, just that this will occur eventually. This is essentially "next value" prediction as characterised by Barzdin [1], which follows from Gold's notion of identifiability in the limit for languages [5].

Let $\mathcal{P}(\omega)$ be the set of all predictors able to learn to predict ω. Similarly for sets of sequences $\mathcal{S} \subset \mathbb{B}^\infty$, define $\mathcal{P}(\mathcal{S}) := \bigcap_{\omega \in \mathcal{S}} \mathcal{P}(\omega)$.

The standard measure of complexity for a sequence ω is its Kolmogorov complexity defined by $K(\omega) := \min_{q \in \mathbb{B}^*} \{|q| : \mathcal{U}(q) = \omega\}$. If no such q exists, we define $K(\omega) := \infty$. It can be shown that this measure of complexity depends on our choice of universal Turing machine $\mathcal{U}$, but only up to an additive constant that is independent of ω. As many of our results have this property, we will define $f(x) \overset{+}{<} g(x)$ to mean that $\exists c \in \mathbb{R}, \forall x : f(x) < g(x) + c$. In essentially the same way we can define the Kolmogorov complexity of a string $x \in \mathbb{B}^n$ by requiring that $\mathcal{U}(q)$ halts after generating x. For an extensive treatment of Kolmogorov complexity and its applications see [7] or [2].

3 Prediction of computable sequences

It is easily seen that every computable sequence can be predicted by at least one predictor, and that this predictor need not be significantly more complex than the sequence to be predicted. Unfortunately however, no universal predictor for computable sequences exists, indeed for every predictor there exists a computable sequence which it cannot predict at all:

Lemma 1. *For any predictor p there constructively exists a sequence $\omega := x_1 x_2 \ldots \in \mathcal{C}$ such that $\forall n \in \mathbb{N} : p(x_{1:n}) \neq x_{n+1}$ and $K(\omega) \overset{+}{<} K(p)$.*

Proof. For any computable predictor p there constructively exists a computable sequence $\omega = x_1 x_2 x_3 \ldots$ computed by an algorithm q defined as follows: Set $x_1 = 1 - p(\lambda)$, then $x_2 = 1 - p(x_1)$, then $x_3 = 1 - p(x_{1:2})$ and so on. Clearly $\omega \in \mathcal{C}$ and $\forall n \in \mathbb{N} : p(x_{1:n}) = 1 - x_{n+1}$.

Let p^* be the shortest program that computes the same function as p and define a sequence generation algorithm q^* based on p^* using the procedure above. By construction, $|q^*| = |p^*| + c$ for some constant c that is independent of p^*. Because q^* generates ω, it follows that $K(\omega) \leq |q^*|$. By definition $K(p) = |p^*|$ and so $K(\omega) \overset{+}{<} K(p)$. $\square$

As the computable prediction of any computable sequence is impossible, a weaker goal is to be able to predict all "simple" computable sequences.

Definition 4. *For $n \in \mathbb{N}$, let $\mathcal{C}_n := \{\omega \in \mathcal{C} : K(\omega) \leq n\}$. Further, let $\mathcal{P}_n := \mathcal{P}(\mathcal{C}_n)$ be the set of predictors able to learn to predict all sequences in $\mathcal{C}_n$.*

Firstly we note that prediction algorithms exist that can learn to predict all sequences up to a given complexity, and that these predictors need not be significantly more complex than the sequences they can predict:

Lemma 2. $\forall n \in \mathbb{N}, \exists p \in \mathcal{P}_n : K(p) \overset{+}{<} n + O(\log n)$.

Proof. Let $h \in \mathbb{N}$ be the number of programs of length n or less which generate infinite sequences. Build the value of h into a prediction algorithm p constructed as follows:

In the k^{th} prediction cycle run in parallel all programs of length n or less until h of these programs have each produced $k+1$ symbols of output. Next predict according to the $k+1^{th}$ symbol of the generated string whose first k symbols is consistent with the observed string. If two generated strings are consistent with the observed sequence (there cannot be more than two as the strings are binary and have length $k+1$), pick the one which was generated by the program that occurs first in a lexicographical ordering of the programs. If no generated output is consistent, give up and output a fixed symbol.

For sufficiently large k, only the h programs which produce infinite sequences will produce output strings of length k. As this set of sequences is finite, they can be uniquely identified by finite initial strings. Thus for sufficiently large k the predictor p will correctly predict any computable sequence ω for which $K(\omega) \leq n$, that is, $p \in P_n$.

As there are $2^{n+1} - 1$ strings of length n or less, and $h < 2^{n+1}$, we can encode h with $\log_2 h + 2 \log_2 \log_2 h + 1 = n + 2 + 2 \log_2(n+1)$ bits. Thus, $K(p) < n + 2 + 2\log_2(n+1) + c$ for some constant c that is independent of n. □

Unfortunately these powerful predictors are necessarily complex:

Theorem 1. $\forall n \in \mathbb{N} : p \in \mathcal{P}_n \Rightarrow K(p) \overset{+}{>} n$.

Proof. For any $n \in \mathbb{N}$ let $p \in \mathcal{P}_n$, that is, $\forall \omega \in \mathcal{C}_n : p \in \mathcal{P}(\omega)$. By Lemma 1 we know that $\exists\, \omega' \in \mathcal{C} : p \notin \mathcal{P}(\omega')$. As $p \notin \mathcal{P}(\omega')$ it must be the case that $\omega' \notin \mathcal{C}_n$, that is, $K(\omega') \geq n$. From Lemma 1 we also know that $K(p) \overset{+}{>} K(\omega')$ and so the result follows. □

Thus, even though we have made the generous assumption of unlimited computational resources and data to learn from, only complex algorithms can be truly powerful predictors. Naturally, highly complex predictors will be difficult to mathematically analyse. Interestingly, an even stronger result in this direction can be proven showing that beyond some point the mathematical analysis is impossible, even in theory:

Theorem 2. *In any consistent formal axiomatic system $\mathcal{F}$ that is sufficiently rich to express statements of the form "$p \in \mathcal{P}_n$", there exists $m \in \mathbb{N}$ such that for all $n > m$ and for all predictors $p \in \mathcal{P}_n$ the true statement "$p \in \mathcal{P}_n$" cannot be proven in $\mathcal{F}$.*

In other words, even though we have proven that very powerful sequence prediction algorithms exist, beyond a certain complexity it is impossible to find any of these algorithms using mathematics. The proof has a similar structure to Chaitin's information theoretic proof [3] of Gödel incompleteness theorem for formal axiomatic systems [4].

Proof. For each $n \in \mathbb{N}$ let T_n be the set of statements expressed in the formal system $\mathcal{F}$ of the form "$p \in \mathcal{P}_n$", where p is filled in with the complete description of some algorithm in each case. As the set of programs is denumerable, T_n is also denumerable and each element of T_n has finite length. From Lemma 2 and Theorem 1 it follows that each T_n contains infinitely many statements of the form "$p \in \mathcal{P}_n$" which are true.

Fix n and create a search algorithm s that enumerates all proofs in the formal system $\mathcal{F}$ searching for a proof of a statement in the set T_n. As the set T_n is recursive, s can always recognise a proof of a statement in T_n. If s finds any such proof, it outputs the corresponding program p and then halts.

By way of contradiction, assume that s halts, that is, a proof of a theorem in T_n is found and p such that $p \in \mathcal{P}_n$ is generated as output. The size of the algorithm s is a constant (a description of the formal system $\mathcal{F}$ and some proof enumeration code) as well as an $O(\log n)$ term needed to describe n. It follows then that $K(p) \overset{+}{<} O(\log n)$. However from Theorem 1 we know that $K(p) \overset{+}{>} n$. Thus, for sufficiently large n, we have a contradiction and so our assumption of the existence of a proof must be false. That is, for sufficiently large n and for all $p \in \mathcal{P}_n$, the true statement "$p \in \mathcal{P}_n$" cannot be proven within the formal system $\mathcal{F}$. □

The exact value of m depends on our choice of formal system $\mathcal{F}$ and which reference machine $\mathcal{U}$ we use, however for reasonable choices of these the value of m would be in the order of 1000. That is, the bound m is certainly not so large as to be vacuous.

These results can be extended to more general settings, specifically to those problems which are equivalent to, or depend on, sequence prediction. Consider, for example, a reinforcement learning agent interacting with an environment [9, 6]. In each interaction cycle the agent must choose its actions so as to maximise the future rewards that it receives from the environment. Of course the agent cannot know for certain whether or not some action will lead to rewards in the future, thus it must predict these. Clearly, at the heart of reinforcement learning lies a prediction problem, and so the results for computable predictors presented in this paper also apply to computable reinforcement learners. More specifically, from Theorem 1 it follows that very powerful computable reinforcement learners are necessarily complex, and from Theorem 2 it follows that it is impossible to discover extremely powerful reinforcement learning algorithms mathematically.

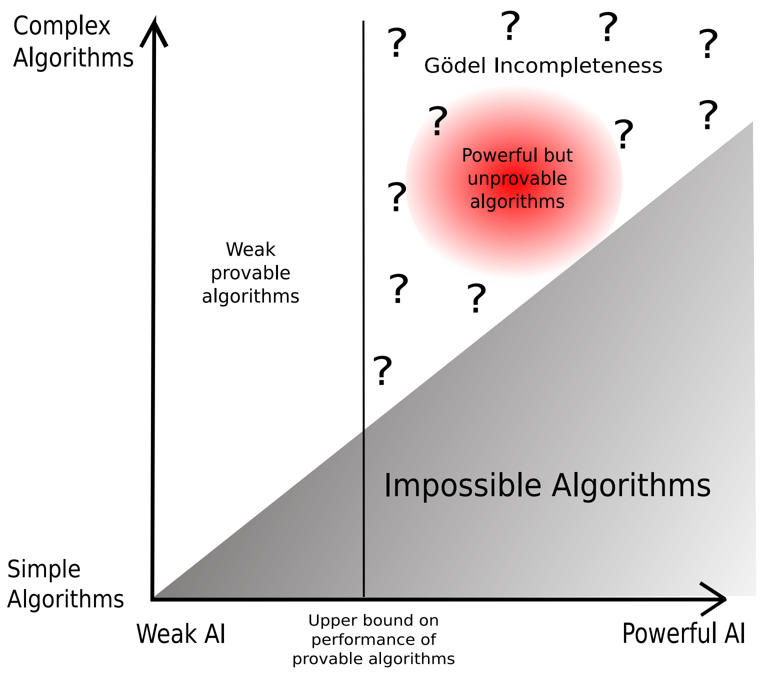

Fig. 1. The results can be depicted as follows. Theorem 1 rules out simple but powerful artificial intelligence algorithms, as indicated by the greyed out region on the lower right. Theorem 2 upper bounds how powerful an algorithm can be before it can no longer be proven to be a powerful algorithm. This is indicated by the vertical line separating the region of provable algorithms from the region of Gödel incompleteness.

References

1. J. M. Barzdin. Prognostication of automata and functions. *Information Processing*, 71:81–84, 1972.
2. C. S. Calude. *Information and Randomness*. Springer, Berlin, 2nd edition, 2002.
3. G. J. Chaitin. Gödel's theorem and information. *International Journal of Theoretical Physics*, 22:941–954, 1982.
4. K. Gödel. Über formal unentscheidbare Sätze der principia mathematica und verwandter systeme I. *Monatshefte für Matematik und Physik*, 38:173–198, 1931. [English translation by E. Mendelsohn: "On undecidable propositions of formal mathematical systems". In M. Davis, editor, *The undecidable*, pages 39–71, New York, 1965. Raven Press, Hewlitt].
5. E. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
6. M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, Berlin, 2004. 300 pages, http://www.idsia.ch/~marcus/ai/uaibook.htm.
7. M. Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Springer, 2nd edition, 1997.
8. R. Penrose. *The Emperor's New Mind.* Oxford U. P., 1989.
9. R. Sutton and A. Barto. *Reinforcement learning: An introduction.* Cambridge, MA, MIT Press, 1998.