

LISP PROGRAM-SIZE COMPLEXITY II

Applied Mathematics and Computation
52 (1992), pp. 103–126

G. J. Chaitin

Abstract

We present the information-theoretic incompleteness theorems that arise in a theory of program-size complexity based on something close to real LISP. The complexity of a formal axiomatic system is defined to be the minimum size in characters of a LISP definition of the proof-checking function associated with the formal system. Using this concrete and easy to understand definition, we show (a) that it is difficult to exhibit complex S-expressions, and (b) that it is difficult to determine the bits of the LISP halting probability Ω_{LISP} . We also construct improved versions Ω'_{LISP} and Ω''_{LISP} of the LISP halting probability that asymptotically have maximum possible LISP complexity.

Copyright © 1992, Elsevier Science Publishing Co., Inc., reprinted by permission.

1. Introduction

The main incompleteness theorems of my *Algorithmic Information Theory* monograph [1] are reformulated and proved here using a concrete and easy-to-understand definition of the complexity of a formal axiomatic system based on something close to real LISP [2]. This paper is the sequel to [3], and develops the incompleteness results associated with the theory of LISP program-size complexity presented in [3]. Further papers in this series shall study (1) a parenthesis-free version of LISP, and (2) a character-string oriented version of LISP in which the naturally occurring LISP halting probability asymptotically has maximum possible LISP complexity.

In [4] I present the latest versions of my information-theoretic incompleteness theorems; there the complexity of a formal system is defined in terms of the program-size complexity of enumerating its infinite set of theorems. Here the goal has been to make these incompleteness results accessible to the widest possible public, by formulating them as concretely and in as straight-forward a manner as possible. Instead of the abstract program-size complexity measure used in [4], here we look at the size in characters of programs in what is essentially real LISP. The price we pay is that our results are weaker (but much easier to prove and understand) than those in [4].

This paper may also be contrasted with the chapter on LISP in my monograph [1]. There I use a toy version of LISP in which identifiers (atoms) are only one character long. In future papers I shall present two LISP dialects that allow multiple-character LISP atoms, but which share some of the desirable features of the toy LISP in [1].

From a technical point of view, Sections 7 and 8 are of special interest. There two artificial LISP halting probabilities Ω'_{LISP} and Ω''_{LISP} are constructed that asymptotically have maximum possible LISP complexity.

At this point it is appropriate to recall LEVIN's delightful and unjustly forgotten book [5] on LISP and metamathematics. (LEVIN was one of the coauthors of the original LISP manual [2].)

Before proceeding, let's summarize the results of [3]. Consider an n -bit string s . Its maximum possible LISP complexity is asymptotic to

a real constant β times n :

$$\max_{|s|=n} H_{\text{LISP}}(s) \sim \beta n.$$

Furthermore, most n -bit strings s have close to this maximum possible LISP complexity; such bit strings are “random.” For example, if $H_{\text{LISP}}(s_n) \sim \beta|s_n|$, then as $n \rightarrow \infty$ the ratio of the number of 0’s to the number of 1’s in the bit string s_n tends to the limit unity.

2. Minimal LISP Expressions

Motivation: Imagine a LISP programming class. Suppose the class is given the following homework assignment: find a LISP S-expression for the list of primes less than a thousand. The students might well compete to find the cleverest solution, the smallest LISP S-expression for this list of prime numbers. But one can never be sure that one has found the best solution! As we shall see, here one is up against fundamental limitations on the power of mathematical reasoning! It is this easy to get in trouble!

Consider a minimal-size LISP S-expression p . I.e., p has the value x and no S-expression smaller than p has the same value x . Also, let q be a minimal-size S-expression for p . I.e., the value of q is p and no expression smaller than q has p as value.

$$q \xrightarrow{\text{yields value}} p \xrightarrow{\text{yields value}} x.$$

Consider the following LISP expression:

(quote p)

This expression evaluates to p . This shows that

$$H_{\text{LISP}}(p) \leq |p| + 8.$$

Consider the following LISP expression:

(eval q)

This expression evaluates to x . This shows that

$$|p| = H_{\text{LISP}}(x) \leq |q| + 7 = H_{\text{LISP}}(p) + 7.$$

Hence

$$H_{\text{LISP}}(p) \geq |p| - 7.$$

In summary,

Theorem A: *If p is a minimal expression, it follows that*

$$\left| H_{\text{LISP}}(p) - |p| \right| \leq 8.$$

Hence the assertion that p is a minimal LISP S-expression is also an assertion about p 's LISP complexity. If one can prove that p is a minimal LISP S-expression, one has also shown that $H_{\text{LISP}}(p) \geq |p| - 7$.

Anticipation: Where do we go from here? Minimal LISP S-expressions illustrate perfectly the ideas explored in the next section, Section 3.

The following result is a corollary of my fundamental theorem on the difficulty of establishing lower bounds on complexity (Theorem C in Section 3): *A formal system with LISP complexity n can only enable one to prove that a LISP S-expression p is minimal if p 's size is $< n + c$ characters. This is only possible for finitely many p , because there are only finitely many expressions (minimal or not) of size $< n + c$.*

Conversely by Theorem D in Section 3, *there are formal systems that have LISP complexity $< n + c'$ in which one can determine each minimal LISP expression p up to n characters in size.* (Basically, the axiom that one needs to know is either the LISP S-expression of size $\leq n$ that takes longest to halt, or the number of LISP S-expressions of size $\leq n$ that halt.)

The details and proofs of these assertions are in Section 3.

3. Exhibiting Complex S-Expressions

Recall the standard LISP convention of having a function return `nil` to indicate “no value”; otherwise the return value is the real value wrapped in a pair of parentheses.

We define an n -character formal system to be an n -character **self-contained** LISP function f that given a purported proof p returns `nil` if the proof is invalid and that returns `(t)` where t is the theorem proved if the proof is valid. I.e., $f(p)$ is always defined and

$$f(p) = \begin{cases} \text{nil} & \text{if } p \text{ is an invalid proof;} \\ (t) & \text{if } p \text{ is a valid proof.} \end{cases}$$

Here is an example of a self-contained function definition:

```
(lambda
  (real-argument-of-function)
  ( (lambda
      (main-function
       auxiliary-function-1
       auxiliary-function-2
       auxiliary-function-3
      )
    (main-function real-argument-of-function)
  )
  (quote definition-of-main-function)
  (quote definition-of-auxiliary-function-1)
  (quote definition-of-auxiliary-function-2)
  (quote definition-of-auxiliary-function-3)
)
)
```

We are interested in the theorems of the following form:

```
(is-greater-than
  (lisp-complexity-of (quote x))
  999999
)
```

This is the LISP notation for

$$H_{\text{LISP}}(x) > 999999.$$

The following theorem works because if we are given a LISP program we can determine its size as well as run it.

Theorem B: *An n -character formal system cannot prove that a specific S-expression has LISP complexity $> n + c$ characters.*

Proof: Suppose we are given a proof-checking algorithm q for a formal system. This is a LISP function of one argument, the putative proof. This function q must always return a value, either `nil` signifying that the proof is incorrect, or a list (`t`) consisting of a single element t , the theorem established by the proof. Given a quoted expression (`quote q`) for the definition of this proof-checking function q , we can do the following two things with it:

1. We can determine the size in characters s of the proof-checker q by converting the S-expression q to a character string¹ and then counting the number of elements in the resulting list of characters.² The LISP for doing this is:

$$s = (\text{length}(\text{character-string } q))$$

2. We can use the proof-checker q to check purported proofs p by forming the expression (`q (quote p)`) and then evaluating this expression in a clean environment. The LISP for doing this is:

```
(eval (cons q
           (cons (cons (quote quote)
                       (cons p
                            nil)))
               nil)))
)
```

So we try the given s -character proof checker q on each possible proof p until we find a valid proof p that

¹The LISP interpreter has to be able to convert S-expressions into character strings in order to print them out. So it might as well make the resulting character strings available internally as well as externally, via a `character-string` built-in function. Characters are integers in the range from 0 to $\alpha - 1$, where α is the size of the LISP alphabet, including the two parentheses and the blank.

²Determining the length of a character string, which is just a list of integers, is easily programmed if it is not provided as a built-in function: `(lambda (list) ((lambda (length) (length list)) (quote (lambda (list) (cond ((atom list) 0) (t (plus 1 (length (cdr list))))))))`.

```
(is-greater-than
  (lisp-complexity-of (quote x))
  n
)
```

where the numeral n is $\geq s + k$. I.e., we are searching for a proof that

$$H_{\text{LISP}}(x) > s + k$$

for a specific S-expression x . Then we output the LISP S-expression x and halt. The computation that we have just described in words can be formulated as a LISP expression

```
((lambda(proof-checker lower-bound)... )
  (quote (lambda(purported-proof)... )
     $\underbrace{\hspace{10em}}_{s \text{ characters}}$ 
     $\underbrace{k}_{[\log_{10} k] + 1 \text{ digits}}$ 
  )
)
```

whose size is $s + [\log_{10} k] + c'$ that evaluates to an S-expression x whose LISP character complexity is $> s + k$. Hence

$$s + [\log_{10} k] + c' > s + k.$$

This yields a contradiction for a fixed choice c of k that depends only on c' and not on the particular formal proof checker that we are given.

Q.E.D.

Above we consider an n -character proof-checker or formal system. What if we consider instead a proof-checker whose LISP complexity is n characters? In fact, a slight modification of the above proof shows that

Theorem C: *A formal system with a LISP complexity of n characters cannot prove that a specific S-expression has LISP complexity $> n + c$ characters.*

Proof: The only change is that the very big LISP expression con-

sidered above now becomes:

```

((lambda(proof-checker lower-bound)... )
  (... )
  minimal-size LISP expression that evaluates to the proof-checker lambda-expression
   $\underbrace{k}$ 
   $\lfloor \log_{10} k \rfloor + 1$  digits
)

```

I.e., the proof-checker is no longer given as a quoted expression, but is itself computed. **Q.E.D.**

This theorem is sharp; here is the converse.

Theorem D: *There is a formal system with LISP complexity $< n+c$ that enables us to determine:*

- (a) *which LISP S-expressions have LISP complexity $\geq n$,³ and*
- (b) *the exact LISP complexity of each LISP S-expression with LISP complexity $< n$.⁴*

Proof: Here are two axioms packed full of information from which we can deduce the desired theorems:

1. Being given the $\leq n$ character LISP S-expression that halts and takes longest to do so (padded to size n).
2. Being given the k th $\leq n$ character LISP S-expression.⁵ Here k is the number of $\leq n$ character LISP S-expressions that halt. (Here again, this S-expression must be padded to a fixed size of n characters.)

Here is how to do the padding:

```
(quote(...)xxxxxx)
```

³There are infinitely many S-expressions with this property.

⁴There are only finitely many S-expressions with this property.

⁵Pick a fixed ordering of all S-expressions, first by size, then alphabetically among S-expressions of the same size.

The three dots are where the S-expression to be padded is inserted. The x 's are the padding. This scheme pads an S-expression e that is $\leq n$ characters long into one $((e)xxxxxx)$ that is exactly $n + 4$ characters long. (The 4 is the number of added parentheses that glue the expression e to its padding $xxxxxx$.) To retrieve the original expression one takes the **CAR** of the **CAR**, i.e., the first element of the first element. To determine n , one converts the padded S-expression into the corresponding character string, one counts the number of characters in it, and one subtracts 4. **Q.E.D.**

4. The Halting Probability Ω_{LISP}

The first step in constructing an expression for a halting probability for real LISP, is to throw out all atomic S-expressions, S-expressions like **harold**, **big-atom**, etc. Anyway, most atomic S-expressions fail to halt in the sense that they fail to have a value. (Of course, this is when they are considered as self-contained S-expressions, not when they are encountered while evaluating a much larger S-expression with lambda-expressions and bindings.) The usual exceptions are the logical constants **t** and **nil**, which evaluate to themselves.⁶ At any rate, the purpose of a halting probability is to help us to decide which S-expressions halt, i.e., have a value. But we don't need any help to decide if an atomic S-expression has a value; this is trivial to do. So let's forget about atomic S-expressions for the moment.

Let's look at all non-atomic S-expressions e , in other words, at S-expressions of the form $(. . .)$. None of these is an extension of another, because the $()$'s must balance and therefore enable us to decide where a non-atomic S-expression finishes. In other words, non-atomic LISP S-expressions have the vital property that they are what is referred to as "self-delimiting." In a moment we shall show that this self-delimiting property enables us to define a LISP halting probability as follows:

⁶In the LISP dialect in [1], an unbound atom will evaluate to itself, i.e., act as if it were a constant.

$$\begin{aligned}
\Omega_{\text{LISP}} &= \sum_{\substack{\text{S-expression } e \\ \text{"halts,"} \\ \text{is defined,} \\ \text{has a value.}}} \alpha^{-[\text{size in characters of S-expression } e]} \\
&= \sum_{(x) \text{ has a value.}} \alpha^{-|(x)|}.
\end{aligned}$$

Here α is the number of characters in the LISP alphabet and is assumed to be a power of two. The S-expressions e included in the above sum must not be atomic, **and they must all be different**. If two expressions e and e' differ only in that one contains redundant blanks, then only the one without the redundant blanks is included in the above sum. Similarly, $(())$ and (nil) are equivalent S-expressions, and are only included once in the sum for Ω_{LISP} . This is a straight-forward definition of a LISP halting probability; we shall see in Sections 7 and 8 that it can be improved.

Ω_{LISP} is also considered to be an infinite bit string, the base-two representation of Ω_{LISP} . It is important to pick the base-two representation **with an infinite number of 1s**. I.e., if it should end with 100000..., pick 011111... instead.⁷

It is crucial that the sum for Ω_{LISP} converges; in fact we have

$$0 < \Omega_{\text{LISP}} < 1.$$

Why is this? The basic reason is that non-atomic LISP S-expressions are self-delimiting because their parentheses must balance. Thus no extension of a non-atomic S-expression is a valid non-atomic S-expression. Extra blanks at the end of an S-expression e are not allowed in the sum for Ω_{LISP} !

Here is a geometrical proof that this works. Associate S-expressions with subsets of the interval of unit length consisting of all real numbers r between zero and one. The S-expression e is associated with all those

⁷We shall see that Ω_{LISP} is highly uncomputable and therefore irrational, so this can't actually occur, but **we don't know that yet!**

real numbers r having that string at the beginning of the fractional part of r 's base- α representation:

$$e \xleftrightarrow{\text{is associated with}} \{ \text{the set of all reals of the form } .e\cdots \text{ in radix } \alpha \text{ notation} \}.$$

Then the length of the interval associated with an S-expression e is precisely its probability $\alpha^{-|e|}$. That no extension of a non-atomic S-expression is a valid non-atomic S-expression means that the intervals associated with non-atomic S-expressions do not overlap. Hence the sum of the lengths of these non-overlapping intervals must be less than unity, since they are all inside an interval of unit length. In other words, the total probability that a string of characters picked at random from an alphabet with α characters is a non-atomic S-expression is less than unity, and from these we select those that halt, i.e., evaluate to a value.

Another crucial property of Ω_{LISP} (and of the two halting probabilities Ω'_{LISP} and Ω''_{LISP} that we will construct in Sections 7 and 8) is that it can be calculated in the limit from below. More precisely, Ω_{LISP} can be obtained as the limit from below of a computable monotone increasing sequence⁸ of dyadic rational numbers⁹ Ω_l :

$$\Omega_0 \leq \Omega_1 \leq \Omega_2 \leq \cdots \leq \Omega_{l-1} \leq \Omega_l \rightarrow \Omega_{\text{LISP}}.$$

This is the case because the set of all S-expressions that halt, i.e., that have a LISP value, is recursively enumerable. In other words, we can eventually discover all S-expressions that halt.

Assume one is given the first $n \log_2 \alpha$ bits of Ω_{LISP} . One then starts to enumerate the set of all S-expressions that halt. As soon as one discovers enough of them to account for the first $n \log_2 \alpha$ bits of Ω_{LISP} , one knows that one has all $\leq n$ character non-atomic S-expressions that halt. And there is a trivial algorithm for deciding which $\leq n$ character atomic S-expressions halt. One then calculates the set of all the values of $\leq n$ character S-expressions that halt, and picks an arbitrary S-expression that is not in this set of values. The result is an S-expression with LISP complexity $> n$. Hence the string of the first $n \log_2 \alpha$ bits of Ω_{LISP} must itself have LISP complexity $> n - c$.

⁸I.e., nondecreasing sequence.

⁹I.e., rationals of the form $i/2^j$.

For more details about the process for using a halting probability to solve the halting problem, see the chapter “Chaitin’s Omega” in GARDNER [6], or see [1].

In summary, if one knows the first $n \log_2 \alpha$ bits of Ω_{LISP} , one can determine each LISP S-expression with LISP complexity $\leq n$ characters, and can then produce a LISP S-expression with LISP complexity $> n$ characters. Thus

Theorem E: *The string consisting of the first $n \log_2 \alpha$ bits of Ω_{LISP} has LISP complexity $> n - c$ characters.*

Using our standard technique for showing that it is difficult to exhibit complex S-expressions (Section 3), it follows immediately that

Theorem F: *To be able to prove what are the values of the first $n \log_2 \alpha$ bits of Ω_{LISP} requires a formal system with LISP complexity $> n - c$ characters.*

Proof: Suppose we are given a proof-checking algorithm for a formal system. This is a LISP function of one argument, the putative proof. This function must always return a value, either `nil` signifying that the proof is incorrect, or a list (`t`) consisting of a single element t , the theorem established by the proof. Given a quoted expression for the definition of this proof-checking function, we both know its size in characters s , and we can use it to check purported proofs. So we try it on each possible proof until we find a proof that “The first $(s + k) \log_2 \alpha$ bits of Ω_{LISP} are ...” This would give us a LISP expression with $s + \lceil \log_{10} k \rceil + c'$ characters that evaluates to something with LISP complexity $> s + k - c''$ characters. This yields a contradiction for a fixed choice of k that depends only on c' and c'' and not on the particular formal proof checker that we are given. **Q.E.D.**

5. Diophantine Equations for Ω_{LISP}

Now let’s convert this incompleteness theorem (Theorem F) into one about diophantine equations.

We arithmetize Ω_{LISP} in two diophantine equations: one polynomial [7], the other exponential [8]. As we pointed out in Section 4, Ω_{LISP} can be obtained as the limit from below of a computable monotone

increasing sequence of dyadic rational numbers Ω_l :

$$\Omega_0 \leq \Omega_1 \leq \Omega_2 \leq \cdots \leq \Omega_{l-1} \leq \Omega_l \rightarrow \Omega_{\text{LISP}}.$$

The methods of JONES and MATIJASEVIČ [7, 8, 26] enable one to construct the following:

D_1 : A diophantine equation

$$P(k, l, x_1, x_2, x_3, \dots) = 0$$

that has one or more solutions if the k th bit of Ω_l is a 1, and that has no solutions if the k th bit of Ω_l is a 0.

D_2 : An exponential diophantine equation

$$L(k, l, x_2, x_3, \dots) = R(k, l, x_2, x_3, \dots)$$

that has exactly one solution if the k th bit of Ω_l is a 1, and that has no solutions if the k th bit of Ω_l is a 0.

Since in the limit of large l the k th bit of Ω_l becomes and remains correct, i.e., identical to the k th bit of Ω_{LISP} , it follows immediately that:

P_1 : There are infinitely many values of l for which the diophantine equation

$$P(k, l, x_1, x_2, x_3, \dots) = 0$$

has a solution iff the k th bit of Ω_{LISP} is a 1.

P_2 : The exponential diophantine equation

$$L(k, x_1, x_2, x_3, \dots) = R(k, x_1, x_2, x_3, \dots)$$

has infinitely many solutions iff the k th bit of Ω_{LISP} is a 1.

Consider the following questions:

Q_1 : For a given value of k , are there infinitely many values of l for which the diophantine equation

$$P(k, l, x_1, x_2, x_3, \dots) = 0$$

has a solution?

Q_2 : For a given value of k , does the exponential diophantine equation

$$L(k, x_1, x_2, x_3, \dots) = R(k, x_1, x_2, x_3, \dots)$$

have infinitely many solutions?

As we have seen in Theorem F, to answer the first $n \log_2 \alpha$ of either of these questions requires a formal system with LISP complexity $> n - c$ characters.

In a more abstract setting [4], with diophantine equations constructed from a different halting probability, we can do a lot better. There answering **any** n of these questions requires a formal system whose set of theorems has enumeration complexity $> n - c$ bits.

In Sections 7 and 8 we construct more artificial versions of Ω_{LISP} , Ω'_{LISP} and Ω''_{LISP} , for which we can show that the LISP complexity of the first n bits is asymptotically the maximum possible, βn characters. By using the method presented in this section, we will automatically get from Ω'_{LISP} and Ω''_{LISP} new versions of the diophantine equations D_1 and D_2 , new versions of the questions Q_1 and Q_2 , and new versions of the corresponding incompleteness theorems. But before diving into these more technical matters, it is a good idea to step back and take a look at what has been accomplished so far.

6. Discussion

The spirit of the results in Section 3 (Theorems B and C) is often expressed as follows:

“A set of axioms of complexity N cannot yield a theorem of complexity [substantially] greater than N .”

This way of describing the situation originated in the introductory discussion of my paper [9]:

“The approach of this paper . . . is to measure the power of a set of axioms, to measure the information that it contains. We shall see that there are circumstances in which one only gets out of a set of axioms what one puts in, and in which

it is possible to reason in the following manner. If a set of theorems constitutes t bits of information, and a set of axioms contains less than t bits of information, then it is impossible to deduce these theorems from these axioms.”

This heuristic principle is basically correct, about as correct as any informal explanation of a technical mathematical result can be. But it is useful to point out its limitations.

In fact, **any** set of axioms that yields an infinite set of theorems **must** yield theorems with arbitrarily high complexity! This is true for the trivial reason that there are only finitely many objects of any given complexity. And it is easy to give natural examples. For example, consider the trivial theorems of the form

$$“N + 1 = 1 + N”$$

in which the numeral N is, if converted to base-two, a large random bit string, i.e., one with LISP complexity $\sim \beta \log_2 N$. (This will be the case for most large integers N .) This theorem has, if considered as a character string, essentially the same arbitrarily large complexity that the number N has.

So what is to become of our heuristic principle that

“A set of axioms of complexity N cannot yield a theorem of complexity substantially greater than N ” ???

An improved version of this heuristic principle, which is not really any less powerful than the original one, is this:

“One cannot prove a theorem from a set of axioms that is of greater complexity than the axioms **and know** that one has done this. I.e., one **cannot realize** that a theorem is of substantially greater complexity than the axioms from which it has been deduced, if this should happen to be the case.”

Thus even though most large integers N are random bit strings in base-two and yield arbitrarily complex theorems of the form

$$“N + 1 = 1 + N”,$$

we can never tell **which** N are random and achieve this!

Note that in Section 4 we encountered no difficulty in using our heuristic principle to restrict the ability of formal systems to prove what the value of Ω_{LISP} is; our general method applies quite naturally in this particular case (Theorem F). This example of the application of our heuristic principle shows that the power of this principle is not restricted by the fact that it really only prevents us from proving theorems that are more complex than our axioms if we can realize that the theorems would be more complex than our axioms are.

Perhaps it is better to avoid all these problems and discussions by rephrasing our fundamental principle in the following totally unobjectionable form:

“A set of axioms of complexity N cannot yield a theorem that asserts that a specific object is of complexity substantially greater than N .”

It was removing the words “asserts that a specific object” that yielded the slightly overly-simplified version of the principle that we discussed above:

“A set of axioms of complexity N cannot yield a theorem that [asserts that a specific object] is of complexity substantially greater than N .”

7. A Second “Halting Probability” Ω'_{LISP}

We shall now “normalize” Ω_{LISP} and make its information content “more dense.” The new halting probability Ω'_{LISP} in this section has a simple definition, but the proof that it works is delicate. The halting probability in Section 8, Ω''_{LISP} , has a more complicated definition, but it is much easier to see that it works.

Let’s pick a total recursive function f such that $\sum 2^{-f(n)} \leq 1$ and $f(n) = O(\log n)$. For example, let $f(n) = 2\lceil \log_2 n \rceil + 1$. This works, because

$$\sum_{n=1}^{\infty} 2^{-2\lceil \log_2 n \rceil - 1} \leq \frac{1}{2} \sum_{n=1}^{\infty} 2^{-2\log_2 n} = \frac{1}{2} \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2/6}{2} \approx \frac{1.644934}{2} < 1.$$

Here we have used the fact discovered by EULER¹⁰ that $1 + 1/4 + 1/9 + 1/16 + 1/25 + \dots = \pi^2/6$.

Define a new LISP “halting probability” between zero and one as follows:

$$\Omega'_{\text{LISP}} = \sum_{n=1}^{\infty} \left[\frac{\# \text{ of different } \leq n \text{ character LISP S-expressions that halt}}{2^{\lceil \log_2(\text{total } \# \text{ of different } \leq n \text{ character LISP S-expressions}) \rceil}} \right] 2^{-2^{\lceil \log_2 n \rceil - 1}}.$$

The factor of $2^{-2^{\lceil \log_2 n \rceil - 1}}$ is in order to insure convergence. The denominator of (total # of different $\leq n$ character LISP S-expressions), or S_n , is increased to the next highest power of two, $2^{\lceil \log_2 S_n \rceil}$, so that we are doing very straightforward binary arithmetic to calculate Ω'_{LISP} .

Why is the LISP complexity of the string of the first n bits of Ω'_{LISP} asymptotic to the maximum possible, βn ?

The main reason is this: Consider all n -bit strings s . From [3] we know that

$$\max_{|s|=n} H_{\text{LISP}}(s) \sim \beta n.$$

And below we shall show that it is also the case that

$$n \sim \log_2(\# \text{ of different } \leq \beta n \text{ character LISP S-expressions}).$$

(This is somewhat delicate.)

So just about when the expression for the denominator in Ω'_{LISP} ,

$$“2^{\lceil \log_2(\text{total } \# \text{ of different } \leq \beta n \text{ character LISP S-expressions}) \rceil}”$$

hits 2^n , the numerator,

$$“\# \text{ of different } \leq \beta n \text{ character LISP S-expressions that halt}”$$

will include all minimal LISP expressions for n -bit strings. Thus knowing a string consisting of the first $n + o(n)$ bits of Ω'_{LISP} tells us the LISP complexity of each $\leq n$ bit string. Hence the LISP

¹⁰For EULER’s proof that $1 + 1/4 + 1/9 + 1/16 + 1/25 + \dots = \pi^2/6$, see Section 6 in Chapter II of the first volume of POLYA [10]. (Also see the exercises at the end of Chapter II.)

complexity of the string of the first n bits of Ω'_{LISP} is asymptotic to $\max_{|s|=n} H_{\text{LISP}}(s) \sim \beta n$.

It remains for us to establish the asymptotic expression for the logarithm of S_n , the total number of different $\leq n$ character LISP S-expressions. Let S'_n be the number of different **non-atomic** $\leq n$ character LISP S-expressions. Consider an S-expression that is a list of n elements, each of which is a $\leq k$ character non-atomic S-expression. This shows that we have

$$S'_{nk+2} \geq (S'_k)^n.$$

(The 2 is for the two enclosing parentheses that must be added.) Hence

$$\log_2 S'_{nk+2} \geq n \log_2 S'_k.$$

Dividing through by nk we see that

$$\frac{\log_2 S'_{nk+2}}{nk} \geq \frac{\log_2 S'_k}{k}.$$

From this it is easy to see that

$$\liminf_{n \rightarrow \infty} \frac{\log_2 S'_n}{n} \geq \frac{\log_2 S'_k}{k}.$$

On the other hand, the **total** number S_n of different $\leq n$ character LISP S-expressions satisfies:

$$n \log_2 \alpha \geq \log_2 S_n > \log_2 S'_n.$$

Thus $(\log_2 S'_n)/n$ tends to the limit $\gamma \leq \log_2 \alpha$ from below, where

$$\gamma = \sup_{n \rightarrow \infty} \frac{\log_2 S'_n}{n} \leq \log_2 \alpha.$$

Furthermore, $\gamma \neq 0$. This can be seen by considering those S-expressions that are a list (999...999) containing a single “bignum” or large integer. (For such S-expressions to be different, the first digit must not be 0.) This shows that $9 \times 10^n \leq S'_{n+3}$, and thus $\gamma \geq \log_2 10 > 0$. So the limit γ of $(\log_2 S'_n)/n$ is not equal to zero, and thus $\log_2 S'_n$ is asymptotic to γn :

$$\log_2 S'_n \sim \gamma n.$$

Consider an S-expression that is a list (...) with one element, which may be an atom. This shows that $S_n \leq S'_{n+2}$. On the other hand, $S'_n \leq S_n$, because each S-expression included in S'_n is also included in S_n . Thus we see that

$$S'_n \leq S_n \leq S'_{n+2}.$$

Since $\log_2 S'_n$ is asymptotic to γn , it follows from this inequality that $\log_2 S_n$ is also asymptotic to γn .

So we have shown that

$$\gamma n \sim \log_2(\# \text{ of different } \leq n \text{ character LISP S-expressions})$$

and therefore

$$n \sim \log_2(\# \text{ of different } \leq n/\gamma \text{ character LISP S-expressions}).$$

We finish by showing that $\beta = 1/\gamma$ by using reasoning from the Appendix of [3]. Order the LISP S-expressions, first by their size in characters, and among those of the same size, in an arbitrary alphabetical order.

To get all n -bit strings, one needs to evaluate at least 2^n different LISP S-expressions. Thus if $S_m < 2^n$, then there is an n -bit string s with LISP complexity greater than m . It follows that $\max_{|s|=n} H_{\text{LISP}}(s) > n/\gamma + o(n)$.

On the other hand, we can use the k th S-expression as a notation to represent the k th bit string. This gives us all n -bit strings by the time k reaches 2^{n+1} . And we have to add c characters to indicate how to convert the k th S-expression into the k th bit string. Thus if $S_m \geq 2^{n+1}$, then all n -bit strings s have LISP complexity less than $m + c$. It follows that $\max_{|s|=n} H_{\text{LISP}}(s) < n/\gamma + o(n)$.

So

$$\max_{|s|=n} H_{\text{LISP}}(s) \sim n/\gamma \sim \beta n,$$

and $\beta = 1/\gamma$. That concludes the proof of the following theorem.

Theorem G: *The LISP complexity of the string consisting of the first n bits of Ω'_{LISP} is $\sim \beta n$. In order to answer the first n questions Q_1 or Q_2 for diophantine equations D_1 and D_2 constructed from Ω'_{LISP} as indicated in Section 5, one needs a formal system with LISP complexity $> \beta n + o(n)$.*

8. A Third “Halting Probability” Ω''_{LISP}

This time the definition of the “halting probability” is more artificial, but it is much easier to see that the definition does what we want it to do.

As in Section 7, we use the fact that

$$\sum_{i=1}^{\infty} 2^{-2^{\lceil \log_2 i \rceil - 1}} \leq \frac{1}{2} \sum_{i=1}^{\infty} 2^{-2 \log_2 i} = \frac{1}{2} \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{12} < 1.$$

Multiplying together two copies of this infinite series, we see that

$$\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} 2^{-2^{\lceil \log_2 i \rceil - 2} - 2^{\lceil \log_2 j \rceil - 2}} < 1^2 = 1.$$

Define a new LISP “halting probability” as follows:

$$\Omega''_{\text{LISP}} = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_{ij} 2^{-2^{\lceil \log_2 i \rceil - 2} - 2^{\lceil \log_2 j \rceil - 2}}.$$

Here the dyadic rationals α_{ij} , for which it is always the case that

$$0 \leq \alpha_{ij} \leq 1,$$

are defined as follows:

$$\alpha_{ij} = \left(\frac{\# \text{ of } j\text{-bit strings that have LISP complexity } \leq i}{\# \text{ of } j\text{-bit strings}} \right) \leq 2^i / 2^j.$$

It follows immediately that

$$0 \leq \Omega''_{\text{LISP}} \leq 1.$$

A string consisting of the first $n + O(\log n)$ bits of Ω''_{LISP} tells us the LISP complexity of each $\leq n$ bit string; the most complex have LISP complexity $\sim \beta n$. So

Theorem H: *The LISP complexity of the string consisting of the first n bits of Ω''_{LISP} is $\sim \beta n$. In order to answer the first n questions Q_1 or Q_2 for diophantine equations D_1 and D_2 constructed from Ω''_{LISP} as indicated in Section 5, one needs a formal system with LISP complexity $> \beta n + o(n)$.*

9. Unpredictability

From the fact that the initial segments of the infinite bit strings Ω'_{LISP} and Ω''_{LISP} asymptotically have maximum possible LISP complexity, it follows that their successive bits cannot be predicted using any computable prediction scheme. More precisely,

Theorem I: *Consider a total recursive prediction function F , which given an arbitrary finite initial segment of an infinite bit string, returns either “no prediction”, “the next bit is a 0”, or “the next bit is a 1”. Then if F predicts at least a fixed nonzero fraction of the bits of Ω'_{LISP} and Ω''_{LISP} , F does no better than chance, because in the limit the relative frequency of correct and incorrect predictions both tend to $\frac{1}{2}$.*

Proof Sketch: We know that Ω'_{LISP} and Ω''_{LISP} both have the property that the string Ω_n of the first n bits of each has LISP complexity asymptotic to the maximum possible, which is βn .

The idea is to separate the n -bit string Ω_n consisting of the first n bits of either Ω'_{LISP} or Ω''_{LISP} into the substring that is not predicted, which we will leave “as is,” and the substring that is predicted, which we will attempt to compress.

Let k be the number of bits of Ω_n that are predicted by F . The $(n - k)$ -bit unpredicted substring of Ω_n we are given “as is.” This takes $\sim \beta(n - k)$ LISP characters.

The k -bit predicted substring of Ω_n is not given directly. Instead, we calculate the predictions made by F , and are given a k -bit string telling us which predictions are correct. Let l be the number of bits that F predicts correctly. Thus this k -bit string will have l 1 bits, indicating “correct,” and $(k - l)$ 0 bits, indicating “incorrect.” If l is not about one-half of k , the string of successes and failures of the prediction scheme will be compressible, from the maximum possible of $\sim \beta k$ LISP characters, to only about $\beta k H(\frac{l}{k}, 1 - \frac{l}{k})$ LISP characters. Here $H(p, q) = -p \log_2 p - q \log_2 q$ is the Boltzmann-Shannon entropy function. $H(p, q)$ is less than one if p and q are not both equal to a half. (For more details, see [3, Section 10].)

In summary, we use the prediction function F to stitch together the unpredicted substring of Ω_n with the predictions. And we are given a compressed string indicating when the predictions are incorrect.

So we have compressed the n -bit string Ω_n into two LISP expressions

of size totaling about

$$\beta(n - k) + \beta k H\left(\frac{l}{k}, 1 - \frac{l}{k}\right) \ll \beta(n - k) + \beta k = \beta n.$$

This is substantially less than βn characters, which is impossible, unless $l \approx k/2$. Thus about half the predictions are correct. **Q.E.D.**

Consider an F that always predicts that the next bit of Ω'_{LISP} is a 1. Applying Theorem I, we see that Ω'_{LISP} has the property that 0's and 1's both have limiting relative frequency $\frac{1}{2}$. Next consider an F that predicts that each 0 bit in Ω'_{LISP} is followed by a 1 bit. In the limit this prediction will be right half the time and wrong half the time. Thus 0 bits are followed by 0 bits half the time, and by 1 bits half the time. It follows by induction that each of the 2^k possible blocks of k bits in Ω'_{LISP} has limiting relative frequency 2^{-k} . Thus, to use BOREL's terminology, Ω'_{LISP} is "normal" in base two; so is Ω''_{LISP} . In fact, Ω'_{LISP} and Ω''_{LISP} are BOREL normal in every base, not just base two; we omit the details.

10. Hilbert's 10th Problem

I would now like to discuss HILBERT's tenth problem in the light of the theory of LISP program-size complexity. I will end with a few controversial remarks about the potential significance of these information-theoretic metamathematical results, and their connection with experimental mathematics and the quasi-empirical school of thought regarding the foundations of mathematics.

Consider a diophantine equation

$$P(k, x_1, x_2, \dots) = 0$$

with parameter k . Ask the question, "Does $P(k) = 0$ have a solution?"

Let

$$q = q_0 q_1 q_2 \dots$$

be the infinite bit string whose k th bit q_k is a 0 if $P(k) = 0$ has no solution, and is a 1 if $P(k) = 0$ has a solution:

$$q_k = \begin{cases} 0 & \text{if } P(k) = 0 \text{ has no solution,} \\ 1 & \text{if } P(k) = 0 \text{ has a solution.} \end{cases}$$

Let

$$q^n = q_0 q_1 \cdots q_{n-1}$$

be the string of the first n bits of the infinite string q , i.e., the string of answers to the first n questions. Consider the LISP complexity of q^n , $H_{\text{LISP}}(q^n)$, the size in characters of the smallest LISP expression whose value is q^n .

If HILBERT had been right and every mathematical question had a solution, then there would be a finite set of axioms from which one could deduce whether $P(k) = 0$ has a solution or not for each k . We would then have

$$H_{\text{LISP}}(q^n) \leq H_{\text{LISP}}(n) + c.$$

The c characters are the finite amount of LISP complexity in our axioms, and this inequality asserts that if one is given n , using the axioms one can compute q^n , i.e., decide which among the first n cases of the diophantine equation have solutions and which don't. Thus we would have

$$H_{\text{LISP}}(q^n) \leq \lfloor \log_{10} n \rfloor + 1 + c = O(\log n).$$

($\lfloor \log_{10} n \rfloor + 1$ is the number of digits in the base-ten numeral for n .) I.e., the LISP complexity $H_{\text{LISP}}(q^n)$ of answering the first n questions would be at most order of $\log n$ characters. We ignore the immense amount of **time** it might take to deduce the answers from the axioms; we are concentrating instead on the **size in characters** of the LISP expressions that are involved.

In 1970 MATIJASEVIČ (see [7]) showed that there is no algorithm for deciding if a diophantine equation can be solved. However, if we are told the number m of equations $P(k) = 0$ with $k < n$ that have a solution, then we can eventually determine which do and which don't. This shows that

$$H_{\text{LISP}}(q^n) \leq H_{\text{LISP}}(n) + H_{\text{LISP}}(m) + c'$$

for some $m \leq n$, which implies that

$$H_{\text{LISP}}(q^n) \leq 2(\lfloor \log_{10} n \rfloor + 1) + c' = O(\log n).$$

I.e., the LISP complexity $H_{\text{LISP}}(q^n)$ of answering the first n questions is **still** at most order of $\log n$ characters. So from the point of view

of the LISP theory of program size, HILBERT's tenth problem, while undecidable, does not look too difficult.

Using the method we presented in Section 5, one can use the "improved" LISP halting probability Ω'_{LISP} or Ω''_{LISP} of Sections 7 and 8 to construct an exponential diophantine equation

$$L(k, x_1, x_2, \dots) = R(k, x_1, x_2, \dots)$$

with a parameter k . This equation yields randomness and unpredictability as follows. Ask the question, "Does $L(k) = R(k)$ have infinitely many solutions?" Now let

$$q = q_0q_1q_2 \cdots$$

be the infinite bit string whose k th bit q_k is a 0 if $L(k) = R(k)$ has finitely many solutions, and is a 1 if $L(k) = R(k)$ has infinitely many solutions:

$$q_k = \begin{cases} 0 & \text{if } L(k) = R(k) \text{ has finitely many solutions,} \\ 1 & \text{if } L(k) = R(k) \text{ has infinitely many solutions.} \end{cases}$$

As before, let

$$q^n = q_0q_1 \cdots q_{n-1}$$

be the string of the first n bits of the infinite string q , i.e., the string of answers to the first n questions. Consider the LISP complexity of q^n , $H_{\text{LISP}}(q^n)$, the size in characters of the smallest LISP expression whose value is q^n . Now we have

$$H_{\text{LISP}}(q^n) \sim \beta n,$$

i.e., the string of answers to the first n questions q^n has a LISP complexity that is asymptotic to the maximum possible for an n -bit string. As we discussed in Section 9, it follows that the string of answers $q = q_0q_1q_2 \cdots$ is now **algorithmically random**, in the sense that any computable prediction scheme that predicts at least a fixed nonzero fraction of the bits of q^n will do no better than chance.¹¹

¹¹In the limit exactly half the predictions will be correct and half the predictions will be incorrect.

Surprisingly, HILBERT was wrong to assume that every mathematical question has a solution. The above exponential diophantine equation yields an infinite series of mathematical facts having maximum possible LISP complexity, asymptotically β LISP characters per yes/no fact. It yields an infinite series of questions which reasoning is powerless to answer because their infinite LISP complexity exceeds the finite LISP complexity of any finite set of mathematical axioms! Here one can get out as theorems only as much LISP complexity as one explicitly puts in as axioms, and reasoning is completely useless! I think this approach to incompleteness via program-size complexity makes incompleteness look much more natural and pervasive than has previously been the case. This new approach also provides some theoretical justification for the experimental mathematics made possible by the computer, and for the new quasi-empirical view of the philosophy of mathematics that is displacing the traditional formalist, logicist, and intuitionist positions.

For other discussions of the significance of these information-theoretic incompleteness theorems, see [11–25].

References

- [1] G. J. CHAITIN, *Algorithmic Information Theory*, 3rd Printing, Cambridge: Cambridge University Press (1990).
- [2] J. MCCARTHY et al., *LISP 1.5 Programmer's Manual*, Cambridge MA: MIT Press (1962).
- [3] G. J. CHAITIN, "LISP program-size complexity," *Applied Mathematics and Computation* **49** (1992), 79–93.
- [4] G. J. CHAITIN, "Information-theoretic incompleteness," *Applied Mathematics and Computation*, in press.
- [5] M. LEVIN, *Mathematical Logic for Computer Scientists*, Report TR-131, Cambridge MA: MIT Project MAC (1974).
- [6] M. GARDNER, *Fractal Music, Hypercards and More...*, New York: Freeman (1992).

- [7] J. P. JONES and Y. V. MATIJASEVIČ, “Proof of the recursive unsolvability of Hilbert’s tenth problem,” *American Mathematical Monthly* **98** (1991), 689–709.
- [8] J. P. JONES and Y. V. MATIJASEVIČ, “Register machine proof of the theorem on exponential diophantine representation of enumerable sets,” *Journal of Symbolic Logic* **49** (1984), 818–829.
- [9] G. J. CHAITIN, “Information-theoretic limitations of formal systems,” *Journal of the ACM* **21** (1974), 403–424.
- [10] G. POLYA, *Mathematics and Plausible Reasoning*, Princeton: Princeton University Press (1990).
- [11] G. J. CHAITIN, “Randomness and mathematical proof,” *Scientific American* **232**:5 (1975), 47–52.
- [12] G. J. CHAITIN, “Gödel’s theorem and information,” *International Journal of Theoretical Physics* **22** (1982), 941–954.
- [13] G. J. CHAITIN, “Randomness in arithmetic,” *Scientific American* **259**:1 (1988), 80–85.
- [14] G. J. CHAITIN, “Undecidability and randomness in pure mathematics,” (transcript of a lecture delivered 28 September 1989 at a SOLVAY conference in Brussels). In: G. J. CHAITIN, *Information, Randomness & Incompleteness—Papers on Algorithmic Information Theory*, 2nd Edition, Singapore: World Scientific (1990), 307–313.
- [15] G. J. CHAITIN, “A random walk in arithmetic,” *New Scientist* **125**:1709 (1990), 44–46. Reprinted in: N. HALL, *The New Scientist Guide to Chaos*, Harmondsworth: Penguin (1991), 196–202.
- [16] J. L. CASTI, *Searching for Certainty*, New York: Morrow (1990).
- [17] G. J. CHAITIN, “Number and randomness,” (transcript of a lecture delivered 15 January 1991 at the Technical University of Vienna). In: M. E. CARVALLO, *Nature, Cognition and System*, Vol. 3, Dordrecht: Kluwer (1992), in press.

- [18] G. J. CHAITIN, “Le hasard des nombres,” *La Recherche* **22** (1991), 610–615.
- [19] D. RUELLE, *Chance and Chaos*, Princeton: Princeton University Press (1991).
- [20] D. RUELLE, *Hasard et Chaos*, Paris: Odile Jacob (1991).
- [21] L. BRISSON and F. W. MEYERSTEIN, *Inventer L’Univers*, Paris: Les Belles Lettres (1991).
- [22] J. A. PAULOS, *Beyond Numeracy*, New York: Knopf (1991).
- [23] J. D. BARROW, *Theories of Everything*, Oxford: Clarendon Press (1991).
- [24] T. NØRRETRANDERS, *Mærk Verden*, Denmark: Gyldendal (1991).
- [25] P. DAVIES, *The Mind of God*, New York: Simon & Schuster (1992).
- [26] C. SMORYŃSKI, *Logical Number Theory I*, Berlin: Springer-Verlag (1991).