Automatic Complexity

Bjørn Kjos-Hanssen

December 22, 2024

This Lean project contains solutions to the exercises in the book *Automatic complexity: a computable measure of irregularity* published by de Gruyter, and excerpts from the book itself.

Preface

As the 1968 film *2001: A Space Odyssey* gave an enigmatic and scientifically accurate depiction of space flight, so Jeffrey O. Shallit and Ming-Wei Wang's paper *Automatic complexity of strings* [15] from 2001 described what we can call a "state odyssey": journeys through the states of a finite automaton that held the promise of further deep exploration.

While Kolmogorov complexity is only defined "up to an additive constant", automatic complexity gives concrete values. When I start up the Complexity Guessing Game at http://math.hawaii.edu/wordpress/bjoern/software/web/complexity-guessing-game/ on November 14, 2021, I am presented with the string $x = 011000111001010$ of length 15, and asked to guess its complexity, from 1 to 8. As we shall see in this book, in particular in Chapter 5, the best bet is to choose the largest complexity offered (8 in this case) unless you spot something very special about x. This is correct for our x and next I am asked about the length 25 word

$$y = 1011001111101111010100101 \tag{1}$$

and for a guess for its complexity, from 1 to 13. Again I choose the maximum, but in this case, the game responds that the complexity is 12 and that there is a complexity *deficiency* of 1.

The idea of complexity (or randomness) deficiency comes from the study of Kolmogorov complexity. However, automatic complexity is a more manageable (and computable) measure of irregularity. When we say "irregularity" here it is partly a pun: automatic complexity is based on finite automata, that accept regular languages, so irregularity indicates a failure of small finite automata to uniquely identify the string in a sense.

This research area was started by Jeff Shallit and Ming-Wei Wang in 2001 [15]. I independently made the same definition in 2009 while teaching the class Math 301 (Discrete Mathematics) at University of Hawai'i. Subsequently, I have written several papers which are treated in this book. Moreover, Jordon and Moser wrote a paper on the topic in 2021 [6]. It is my hope that this book will stimulate further work in this area.

As a research tool, and incidentally as a method of cheating at the *Complexity Guessing Game*, I have created a web service to find the complexity of a given word, and an illustration of an automaton used in the associated proof [7].

The Complexity Option Game [8] is a variation on the same idea, inviting the player implement an exercise policy for a complexity-based financial option. These games include graphical displays of millions of the relevant automata.

How to use this book. Chapter 1, Chapter 2, Chapter 3, and Chapter 4 answer the question "What" by introducing the basics of state-counting and edge-counting automatic complexity. Chapter 5 answers the question "How" (do we work with automatic complexity), answering a question of Shallit and Wang by estimating the complexity of random words. Chapter 6 and Chapter 7 are an attempt to answer the question "Why". Conditional automatic complexity gives a perspective on the length-conditional aspect of automatic complexity, and automatic

complexity turns out to give an answer to the question, what does logical depth look like in practice.

Each chapter contains exercises, most of which come with solutions in the proof assistant Lean. Open research problems also appear in dedicated sections.

Acknowledgments. I am grateful to many people.

- Andrew J.I. Jones, Dag Normann, Theodore A. Slaman, and many others mentored me in computability and logic.
- In a Discrete Mathematics class in Spring 2009, students Jason Axelson, Chris Ho and Aaron Kondo wrote a C program that calculated the complexity of all strings of length at most 7. The dedication they put into that program fueled my interest in automatic complexity.
- Logan Axon wrote the first Python script for automatic complexity.
- Kayleigh K. Hyde's 2013 Master's project and her proof of the sharp upper bound for nondeterministic automatic complexity sparked my interest in proving theorems in this area.
- Students who have worked with me on automatic complexity include Samuel D. Birns, Calvin K. Bannister, Swarnalakshmi (Janani) Lakshmanan and Daylan K. Yogi.
- Jeff Shallit, Achilles Beros, Nikolai Vereshchagin, Sasha Shen, André Nies, Frank Stephan and Angeliki Koutsoukou-Argyraki provided encouragement and interesting discussions.

This research was supported in part by a grant from Decision Research Corporation (University of Hawai'i Foundation Account #129-4770-4). This work was partially supported by a grant from the Simons Foundation (#704836 to Bjørn Kjos-Hanssen).

While I have tried to keep this book *akamai*, errors may occur and are my responsibility. I would be grateful to receive reports at `bjoernkh+acmoi@hawaii.edu`.

Honolulu, October 2023

Chapter 1

First steps in automatic complexity

The Kolmogorov complexity of a finite word w is, roughly speaking, the length of the shortest description w^* of w in a fixed formal language. The description w^* can be thought of as an optimally compressed version of w. Motivated by the non-computability of Kolmogorov complexity, Shallit and Wang [15] studied a deterministic finite automaton analogue.

Their notion of automatic complexity is an automata-based and length-conditional analogue of Sipser's distinguishing complexity CD ([16], [12, Definition 7.1.4]). This was pointed out by Mia Minnes in a review in the *Bulletin of Symbolic Logic* from 2012. Another precursor is the length-conditional Kolmogorov complexity [12, Definition 2.2.2].

The automatic complexity of Shallit and Wang is the minimal number of states of an automaton accepting only a given word among its equal-length peers. Finding such an automaton is analogous to the protein folding problem where one looks for a minimum-energy configuration. The protein folding problem may be NP-complete [3], depending on how one formalizes it as a mathematical problem. For automatic complexity, the computational complexity is not known, but a certain generalization to equivalence relations gives an NP-complete decision problem [9].

In this chapter we start to develop the properties of automatic complexity.

1.1 Words

The set of all natural numbers is $\mathbb{N} = \{0, 1, 2, \dots\}$. Following the von Neumann convention, each natural number $n \in \mathbb{N}$ is considered to be the set of its predecessors:

$$0 = \emptyset, \quad 1 = \{0\}, \text{ and in general } \quad n = \{0, 1, \dots, n-1\}.$$

The power set $\mathcal{P}(X)$ of a set X is the set of all the subsets of X:

$$\mathcal{P}(X) = \{A \mid A \subseteq X\}.$$

If Σ is an *alphabet* (a set), a *word* (sometimes called *string*) is a sequence of elements of Σ.

For computer implementations it is often convenient to use an alphabet that is an interval $[0, b)$ in $\mathbb{N}$. When we want to emphasize that 0, 1, etc. are playing the role of symbols rather than numbers, we often typeset them as $\mathtt{0}, \mathtt{1}$, etc., respectively.

We denote concatenation of words by $x +\!\!+ y$ or by juxtaposition xy. In the word $w = xyz$, y is called a *subword* or *factor* of w. Infinite words are denoted in boldface. For example, there is a unique infinite word $\mathbf{w}$ such that $\mathbf{w} = 0\mathbf{w}$, and we write $\mathbf{w} = 0^\infty$.

Let $\preceq$ denote the prefix relation, so that $a \preceq b$ iff a is a prefix of b, iff there is a word c such that $b = ac$.

The concatenation of a word x and a symbol a is written $x :; a$ if a is appended on the right, and $a :: x$ if a is appended on the left. It may seem most natural to define Σ^* by induction using $x :; a$, at least for speakers of languages where one reads from left to right. The Lean proof assistant [4] (version 3) uses $a :: x$. That approach fits well with co-induction, if infinite words are ordered in order type $\mathbb{N}$ [10].

For $n \in \mathbb{N}$, Σ^n is the set of words of length n over Σ. We may view $\sigma \in \Sigma^n$ is a function with domain n and range Σ.

We view functions $f : A \to B$ as subsets of the cartesian product $A \times B$,

$$f = \{(x, y) \mid y = f(x)\}.$$

The set Σ^n is both the set of functions from n to Σ and the cartesian product $(\Sigma^{n-1}) \times \Sigma$ if $n > 0$. The empty word is denoted ε and the first symbol in a word x is denoted $x(0)$.

We can define $\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n$. More properly, the set Σ^* is defined recursively by the rule that $\varepsilon \in \Sigma^*$, and whenever $s \in \Sigma^*$ and $a \in \Sigma$, then $s :; a \in \Sigma^*$. We define *concatenation* by structural induction: for $s, t \in \Sigma^*$,

$$\begin{aligned} t +\!\!+ \varepsilon &= t, \\ t +\!\!+ (s :; a) &= (t +\!\!+ s) :; a. \end{aligned}$$

Definition 1. The *length* of a word $s \in \Sigma^*$ is defined by induction:

$$\begin{aligned} |\varepsilon| &= 0 \\ |s :; a| &= |s| + 1. \end{aligned}$$

If $A \subseteq B$ and $f \subseteq B \times C$ then $f \restriction A = \{(x, y) \in f \mid x \in A\}$ is the restriction of f to A.

The word σ is also denoted $\langle \sigma(0), \sigma(1), \ldots, \sigma(|\sigma| - 1)\rangle$. By convention, instead of $\langle \mathtt{0}, \mathtt{1}, \mathtt{0}\rangle$ we write simply $\mathtt{010}$.

Example 2. We have

$$\begin{aligned} \langle \mathtt{0}\rangle +\!\!+ \langle \mathtt{1}, \mathtt{0}\rangle &= \langle \mathtt{0}, \mathtt{1}, \mathtt{0}\rangle \\ &= \mathtt{0} :: \langle 1, 0\rangle \\ &= \langle 0, 1\rangle :; \langle 0\rangle. \end{aligned}$$

1.1.1 Occurrences and powers

In this subsection we state some results from the subject "combinatorics on words" that will be used frequently.

The statement $\mathrm{occurs}(x, k, y)$ that x occurs in position k within y may be defined by induction on n:

$$\begin{aligned} \mathrm{occurs}(x, 0, y) &\iff \exists z,\ x +\!\!+ z = y, \\ \mathrm{occurs}(x, n + 1, y) &\iff \exists a \in \Sigma,\ \mathrm{occurs}(a :: x, n, y). \end{aligned}$$

Example 3. We have occurs(na, 2, banana) and occurs(na, 4, banana).

The number of occurrences can be defined, without defining a notion of "occurrence", as the cardinality of $\{k \in \mathbb{N} : \text{occurs}(x, k, y)\}$. To define disjoint occurrences, so we should have a notion of "occurrence".

Definition 4. Two occurrences of words a (starting at position i) and b (starting at position j) in a word x are *disjoint* if $x = uavbw$ where u, v, w are words and $|u| = i$, $|uav| = j$.

Type-theoretically [2] we may say that an occurrence of x in y is a pair (k, h) where h is a proof that occurs(x, k, y). Of course, we could also say that the occurrence is simply the number k, or even the triple (x, k, y) but in that case the object does not have its defining property within it, so to speak: the number k, and the triple (x, k, y), exist even when x does not occur at position k in y.

To naturally speak of disjoint occurrences we make Definition 5. Note that we primarily use zero-based words in this book, i.e., other things being equal we prefer to call the first letter of a word x_0, rather than x_1.

Definition 5. A word $x = x_0 \dots x_{n-1}$, or an infinite word $x = x_0 x_1 \dots$, with each $x_i \in \Sigma$, is viewed as a function $f_x : n \to \Sigma$ with $f_x(i) = x_i$. An *occurrence* of $y = y_0 \dots y_{m-1}$ in x is a function $f_x \upharpoonright [a, a + m - 1]$ such that $f_x(a + i) = y_i$ for each $0 \leq i < m$.

For $a, b \in \mathbb{N}$, let $[a, b] = \{x \in \mathbb{N} \mid a \leq x \leq b\}$. Two occurrences $f_x \upharpoonright [a, b]$, $f_x \upharpoonright [c, d]$ are *disjoint* if $[a, b] \cap [c, d] = \emptyset$.

If moreover $[a, b + 1] \cap [c, d] = \emptyset$ then the occurrences are *strongly disjoint*.

A subword that occurs at least twice in a word w is a *repeated* subword of w. Let $k \in \mathbb{N}$, $k \geq 1$. A word $x = x_0 \dots x_{n-1}$, $x_i \in \Sigma$, is *k-rainbow* if it has no repeated subword of length k: there are no $0 \leq i < j < n - k$ with $x \upharpoonright [i, i + k - 1] = x \upharpoonright [j, j + k - 1]$. A 1-rainbow word is also known simply as *rainbow*.

In particular, the empty word ε occurs everywhere in every word. However, each word has exactly one occurrence of ε, since all empty functions are considered to be equal (in set theory, at any rate).

Having properly defined "occurrence" in Definition 5, we can state and prove the trivial Lemma 6.

Lemma 6. *Suppose n, t are positive integers with $t \leq n + 1$. A word of length n has $n + 1 - t$ occurrences of subwords of length t.*

Proof. Let x be a word of length n. The occurrences of subwords of length t are

$$f_x \upharpoonright [0, t - 1], f_x \upharpoonright [1, t], \dots, f_x \upharpoonright [n - t, n - t + (t - 1)].$$

□

Theorem 7. *Let $k, t \in \mathbb{N}$ with $k \geq 1$. In an alphabet of cardinality k, a t-rainbow word has length at most $k^t + t - 1$.*

Proof. There are k^t words of length t. Thus, by Lemma 6, for a t-rainbow word of length n we have $n + 1 - t \leq k^t$. □

Theorem 7 has a converse, Theorem 9. To prove it we shall require the notion of a de Bruijn word.

Definition 8. A *de Bruijn word of order n* over an alphabet Σ is a sequence y such that every $x \in \Sigma^n$ occurs exactly once as a cyclic substring of y.

Theorem 9. *Let $k, t \in \mathbb{N}$ with $k \geq 1$. In an alphabet of cardinality k, there exists a t-rainbow word of length $k^t + t - 1$.*

Proof. Case $t = 0$: indeed, the empty word is a 0-rainbow word of length 0. Case $t = 1$: A 1-rainbow word of length k exists, namely any permutation of the symbols in Σ (Definition 5). For $t > 1$, let x be a *de Bruijn* word $B(k, t)$ of length k^t and let $w = x^2 \upharpoonright (k^t + t - 1)$. □

Definition 10. Let α be a word of length n, and let α_i be the i^{th} letter of α for $1 \leq i \leq n$. We define the u^{th} power of α for certain values of $u \in \mathbb{Q}_{\geq 0}$ (the set of nonnegative rational numbers) as follows. For $u \in \mathbb{N}$:

$$\begin{aligned} \alpha^0 &= \varepsilon, \\ \alpha^{n+1} &= \alpha^n +\!\!+\, \alpha. \end{aligned}$$

- If $u = v + k/n$ where $0 < k < n$, and k is an integer, then α^u denotes $\alpha^v \alpha_1 \ldots \alpha_k$ and is called a u-power.

The word w is u-power-free if no nonempty v-power, $v \geq u$, occurs (Definition 5) in w. In particular, 2-power-free is called *square-free* and 3-power-free is called *cube-free*. Let $\mathbf{w}$ be an infinite word over the alphabet Σ, and let x be a finite word over Σ. Let $u > 0$ be a rational number. The word x is said to occur in $\mathbf{w}$ with exponent u if x^u occurs in $\mathbf{w}$ (Definition 5).

The reader may note that the definition of u-power-free is perhaps not obvious. For example, the word *aba* is not 1.49-power-free: while it contains no 1.49-power, it contains a 1.5-power. This way of defining things enables Theorem 12 and goes back at least to Krieger [11, page 71].

Definition 11. The *critical exponent* $\mathrm{ce}(w)$ of an infinite word $\mathbf{w}$ is defined by

$$\mathrm{ce}(w) = \sup\{\alpha \in \mathbb{Q} \mid \mathbf{w} \text{ contains some } \alpha\text{-power}\}.$$

Theorem 12 (Krieger [11])**.** *The critical exponent of $\mathbf{w}$ is equal to*

$$\inf\{\alpha \in \mathbb{Q} \mid \mathbf{w} \textit{ is } \alpha\textit{-power-free}\}.$$

Proof. Let

$$\begin{aligned} S &= \{\alpha \in \mathbb{Q} \mid \mathbf{w} \text{ is } \alpha\text{-power-free}\}, \\ T &= \{\alpha \in \mathbb{Q} \mid \mathbf{w} \text{ contains some } \alpha\text{-power}\}. \end{aligned}$$

Paying careful attention to Definition 10, the word $\mathbf{w}$ is α-power-free iff for all $\beta \geq \alpha$, $\mathbf{w}$ contains no β-power. Therefore, S is upward closed. On the other hand, T is an upward dense subset of the complement of S. Therefore, $\mathrm{ce}(w) = \sup T = \inf S$. □

As an example of Definition 10, we have $\mathtt{0110}^{3/2} = \mathtt{011001}$. Note that the expected Power Rule for Exponents fails for word exponentiation. In general, $(x^a)^b \neq x^{ab}$, for instance

$$(01)^3 = 010101 \neq 010010 = ((01)^{3/2})^2.$$

Fix an alphabet Σ, and let Σ^+ denote the set of nonempty words over Σ.

Lemma 13 (Lyndon and Schützenberger [13]; see [14, Theorem 2.3.2]). *Let $c, d, e \in \Sigma^+$. The equation $cd = de$ holds iff there exist a nonempty word u, a word v, and a natural number p such that $c = uv$, $d = (uv)^p u$, and $e = vu$.*

Theorem 14 (Lyndon and Schützenberger [13]; see[14, Theorem 2.3.3]). *Let $x, y \in \Sigma^+$. Then the following four conditions are equivalent:*

1. *$xy = yx$.*
2. *There exists $z \in \Sigma$ and integers $k, l > 0$ such that $x = z^k$ and $y = z^l$.*
3. *There exist integers $i, j > 0$ such that $x^i = y^j$.*

Chapter 2

Nondeterminism and overlap-free words

In this chapter we develop some properties of nondeterministic automatic complexity. As a corollary we get a strengthening of a result of Shallit and Wang [15] on the complexity of the infinite Thue–Morse word $\mathbf{t}$. Moreover, viewed through an NFA lens we can, in a sense, characterize the complexity of $\mathbf{t}$ exactly. A main technical idea is to extend the following result, which says that not only do squares, cubes and higher powers of a word have low complexity, but a word completely free of such powers must conversely have high complexity.

Chapter 3

Edge complexity and digraphs

3.1 Edge-counting automatic complexity

Chapter 4

The many variants

4.1 Master diagram

Chapter 5

The incompressibility theorem

Shallit and Wang showed that the automatic complexity $A(x)$ satisfies $A(x) \geq n/13$ for almost all $x \in \{0,1\}^n$. They also stated that Holger Petersen had informed them that the constant 13 can be reduced to 7. Here we show that it can be reduced to $2 + \epsilon$ for any $\epsilon > 0$. The result also applies to nondeterministic automatic complexity $A_N(x)$. In that setting the result is tight inasmuch as $A_N(x) \leq n/2 + 1$ for all x.

Chapter 6

Conditional automatic complexity

In this chapter we show that metrics analogous to the Jaccard distance and the Normalized Information Distance can be defined based on conditional nondeterministic automatic complexity A_N. Our work continues the path of Shannon (1950) on entropy metrics and Gács (1974) on symmetry of information among others.

Shallit and Wang (2001) defined the automatic complexity of a word w as, somewhat roughly speaking, the minimum number of states of a finite automaton that accepts w and no other word of length $|w|$. This definition may sound a bit artificial, as it is not clear the length of w is involved in defining the complexity of w. In this chapter we shall see how *conditional* automatic complexity neatly resolves this issue.

6.1 Basics

Chapter 7

Logical depth and automatic complexity

7.1 Introduction

Logical depth was defined by Chaitin in 1977 [1] and further studied by Bennett in 1986. It is essentially the time it takes to verify a witness of Kolmogorov complexity. We demonstrate that for automatic complexity, logical depth arises as the difficulty of determining the unique solvability of certain knapsack problems.

A word is deep not so much if it has no simple description but rather if its simplest description is hard to understand and verify. For example, the task of finding the factorization

$$2001 = 3 \times 23 \times 29$$

is laborious to find, but relatively simple to verify. In that sense, it is not deep. In this chapter we uncover a quite concrete instance of this idea of logical depth. To do so we replace Turing machines by finite automata. In the case of infinite sequences, finite-state depth has been studied by Jordon and Moser [5], but we focus on the concrete world of finite words.

We replace Kolmogorov complexity with *automatic complexity*. The automatic complexity $A(w)$ of a word w was studied by Shallit and Wang (2001) [15]. It is the minimum number of states of a DFA with certain properties and can be defined in a couple of in-equivalent variants.

Now, an NFA with a minimum number of edges accepting the word w and no other word of the same length must of course contain a walk on which w is accepted that *visits every edge* of M. It is thus natural to require w to be the only word of length $|w|$ accepted by M on an edge-covering walk, thus obtaining the edge-covering complexity $E_{Nc}(w)$.

Bibliography

[1] G. J. Chaitin. Algorithmic information theory. *IBM Journal of Research and Development*, 21(4):350–359, 1977.

[2] Thierry Coquand and Gérard Huet. The calculus of constructions. *Inform. and Comput.*, 76(2-3):95–120, 1988.

[3] Pierluigi Crescenzi, Deborah Goldman, Christos Papadimitriou, Antonio Piccolboni, and Mihalis Yannakakis. On the complexity of protein folding. *Journal of Computational Biology : a journal of computational molecular cell biology*, 5:423–65, 02 1998.

[4] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[5] Liam Jordon and Philippe Moser. On the difference between finite-state and pushdown depth. In *SOFSEM 2020: theory and practice of computer science*, volume 12011 of *Lecture Notes in Comput. Sci.*, pages 187–198. Springer, Cham, [2020] ©2020.

[6] Liam Jordon and Philippe Moser. Normal sequences with non-maximal automatic complexity. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 47:1–47:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[7] Bjørn Kjos-Hanssen. Complexity lookup. http://math.hawaii.edu/wordpress/bjoern/complexity-of-0110100110010110/.

[8] Bjørn Kjos-Hanssen. Complexity option game. http://math.hawaii.edu/wordpress/bjoern/complexity-option-game/.

[9] Bjørn Kjos-Hanssen. On the complexity of automatic complexity. *Theory Comput. Syst.*, 61(4):1427–1439, 2017.

[10] Dexter Kozen and Alexandra Silva. Practical coinduction. *Math. Structures Comput. Sci.*, 27(7):1132–1152, 2017.

[11] Dalia Krieger. On critical exponents in fixed points of non-erasing morphisms. *Theoret. Comput. Sci.*, 376(1-2):70–88, 2007.

[12] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Graduate Texts in Computer Science. Springer-Verlag, New York, second edition, 1997.

[13] R. C. Lyndon and M. P. Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J.*, 9:289–298, 1962.

[14] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory.* Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[15] Jeffrey Shallit and Ming-Wei Wang. Automatic complexity of strings. *J. Autom. Lang. Comb.*, 6(4):537–554, 2001. 2nd Workshop on Descriptional Complexity of Automata, Grammars and Related Structures (London, ON, 2000).

[16] Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 330–335, New York, NY, USA, 1983. ACM.