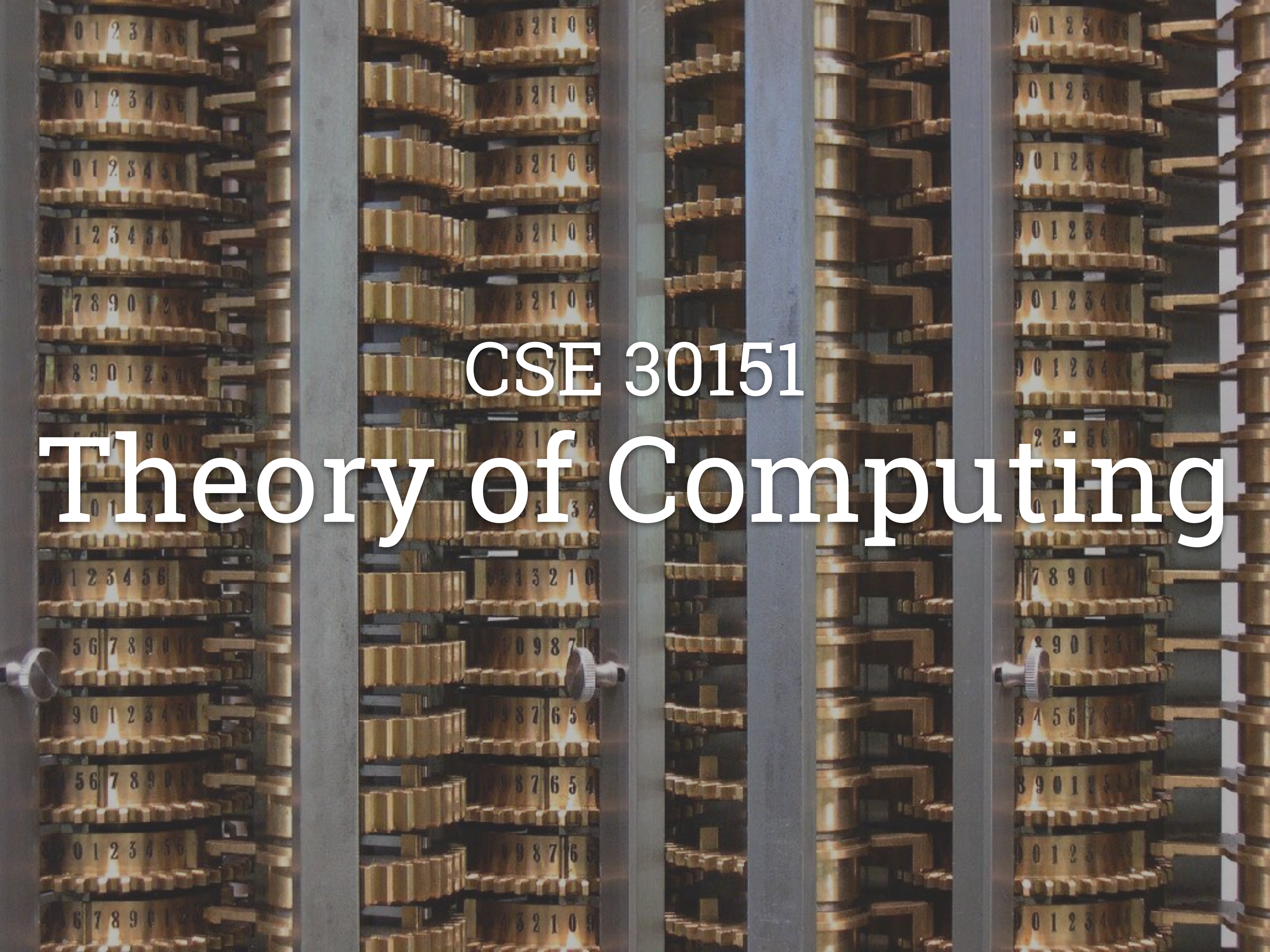


CSE 30151

Theory of Computing

Unit 1–2: Restricted machines

Unit 3: Computability and uncomputability

Unit 4: Tractability and intractability

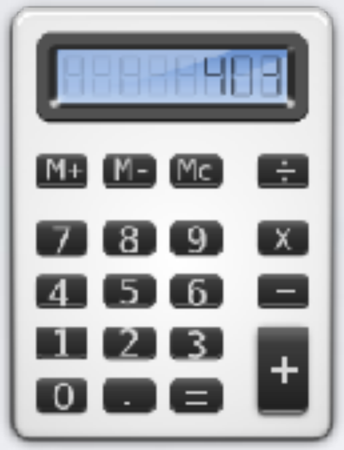
Play

bit.ly/pcpgame

Discuss

How to write a program to solve this puzzle?

WHAT IS A COMPUTER?

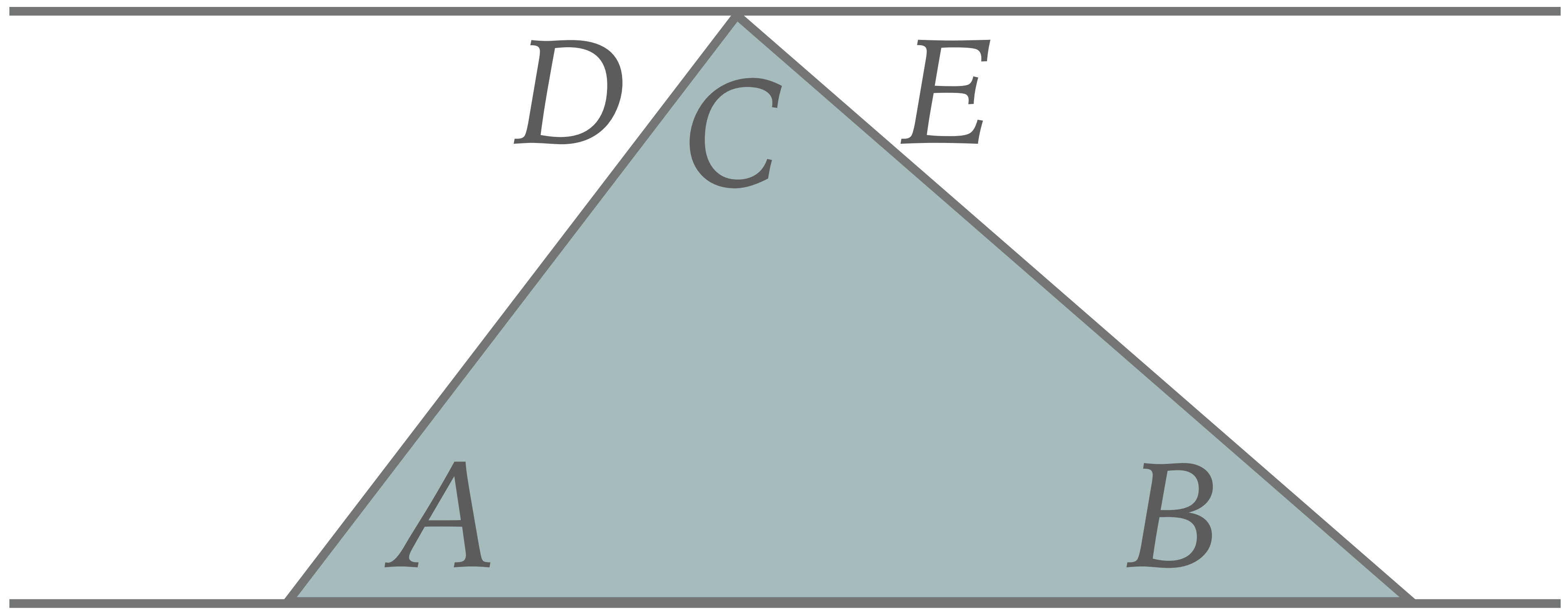


Vote at: piazza.com/nd/spring2018/cse30151

Genius has no race. Strength has no gender.
Courage has no limit.
TARAJI P. HENSON
OCTAVIA SPENCER
JANELLE MONÁE
KEVIN COSTNER
KIRSTEN DUNST
JIM PARSONS
HIDDEN FIGURES
BASED ON THE UNTOLD TRUE STORY
FOX 2000 PICTURES PRESENTS A CHERNIN ENTERTAINMENT/LEVANTINE FILMS PRODUCTION "HIDDEN FIGURES"
TARAJI P. HENSON OCTAVIA SPENCER JANELLE MONÁE KEVIN COSTNER KIRSTEN DUNST JIM PARSONS
HANS ZIMMER, PHARRELL WILLIAMS & BENJAMIN WALLFISCH RENÉE EHRLICH KALFUS PETER TESCHNER WYNN THOMAS
MANDY WALKER, A.S.C. ACS JAMAL DANIEL RENEE WITT IVANA LOMBARDI MIMI VALDÉS KEVIN HALLORAN
DONNA GIGLIOTTI, p.g.a. PETER CHERNIN, p.g.a. JENNO TOPPING, p.g.a. PHARRELL WILLIAMS, p.g.a. THEODORE MELFI, p.g.a.
MARGOT LEE SHETTERLY ALLISON SCHROEDER AND THEODORE MELFI THEODORE MELFI
SOUNDTRACK ON
#HiddenFigures
HiddenFiguresMovie.com
IN CINEMAS FEBRUARY

$$\begin{array}{ccc} & {\scriptstyle 1} & \\ 3 & 1 & 4 \\ 1 & 5 & 9 \\ \hline 4 & 7 & 3 \end{array}$$

$$\begin{aligned} x+2 &= 3x+4 \\ 2 &= 2x+4 \\ -2 &= 2x \\ -1 &= x \end{aligned}$$



$$\angle D = \angle A$$

$$\angle E = \angle B$$

$$\angle D + \angle C + \angle E = 180$$

$$\angle A + \angle B + \angle C = 180$$

FERMAT'S LAST THEOREM: *Let n, a, b, $c \in \mathbf{Z}$ with $n > 2$. If $a^n + b^n = c^n$ then abc $=$ 0.*

Proof: The proof follows a program formulated around 1985 by Frey and Serre [**F,S**]. By classical results of Fermat, Euler, Dirichlet, Legendre, and Lamé, we may assume $n = p$, an odd prime ≥ 11. Suppose $a, b, c \in \mathbf{Z}$, $abc \neq 0$, and $a^p + b^p = c^p$. Without loss of generality we may assume $2|a$ and $b \equiv 1 \bmod 4$. Frey [**F**] observed that the elliptic curve $E : y^2 = x(x - a^p)(x + b^p)$ has the following "remarkable" properties: (1) E is semistable with conductor $N_E = \prod_{\ell|abc} \ell$; and (2) $\bar{\rho}_{E,p}$ is unramified outside $2p$ and is flat at p. By the modularity theorem of Wiles and Taylor-Wiles [**W,T-W**], there is an eigenform $f \in \mathcal{S}_2(\Gamma_0(N_E))$ such that $\rho_{f,p} = \rho_{E,p}$. A theorem of Mazur implies that $\bar{\rho}_{E,p}$ is irreducible, so Ribet's theorem [**R**] produces a Hecke eigenform $g \in \mathcal{S}_2(\Gamma_0(2))$ such that $\rho_{g,p} \equiv \rho_{f,p} \bmod \wp$ for some $\wp|p$. But $X_0(2)$ has genus zero, so $\mathcal{S}_2(\Gamma_0(2)) = 0$. This is a contradiction and Fermat's Last Theorem follows.
Q.E.D.

HILBERT'S ENTSCHEIDUNGSPROBLEM

Given a statement ϕ of first-order logic and some axioms, is there an algorithm that decides whether ϕ is provable from the axioms?

"BENEDICT CUMBERBATCH IS OUTSTANDING"
RADIO TIMES
"THE BEST BRITISH FILM OF THE YEAR"
★★★★★
THE INDEPENDENT
"AN INSTANT CLASSIC"
★★★★★
GLAMOUR
"A SUPERB THRILLER"
★★★★
EMPIRE
★★★★
TIME OUT
★★★★
THE TIMES
BENEDICT CUMBERBATCH
KEIRA KNIGHTLEY
THE IMITATION GAME
12A
BASED ON THE INCREDIBLE TRUE STORY
/ImitationGameUK
IN CINEMAS NOVEMBER 14

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

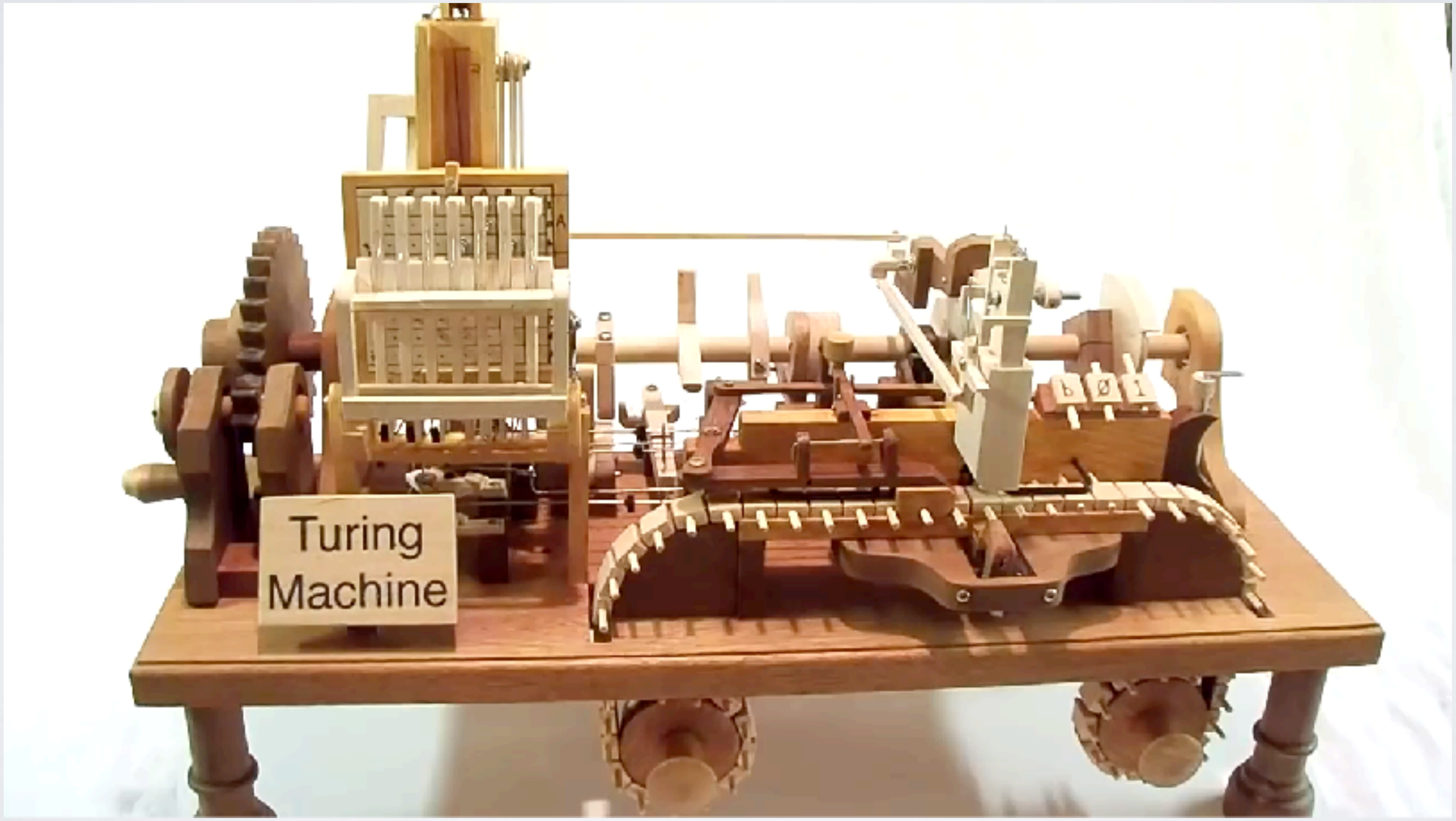
† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

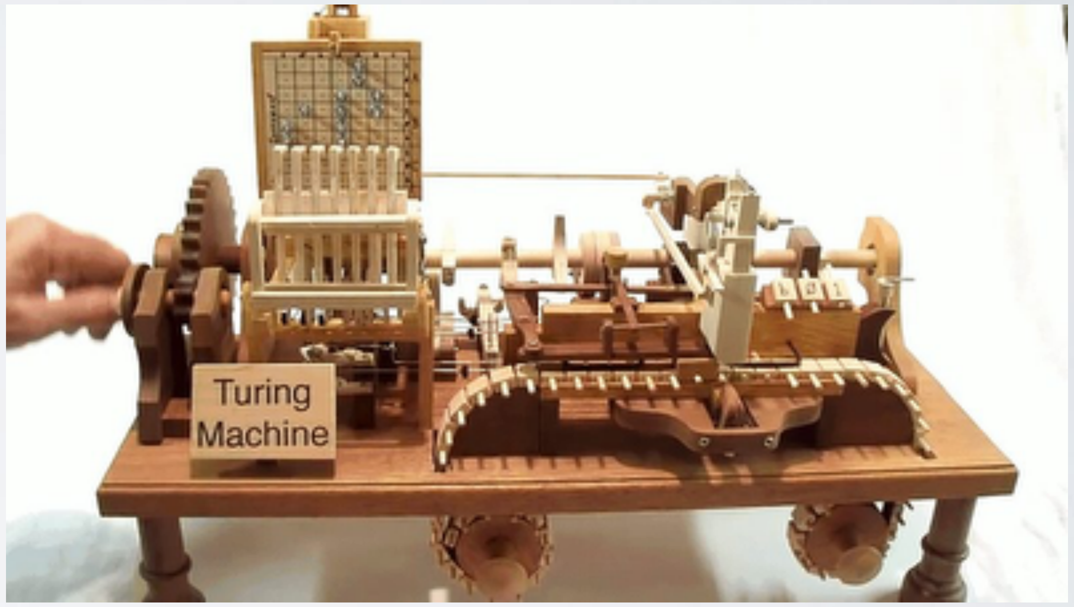
Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable.

TURING MACHINES



https://www.youtube.com/watch?v=vo8izCKHiF0

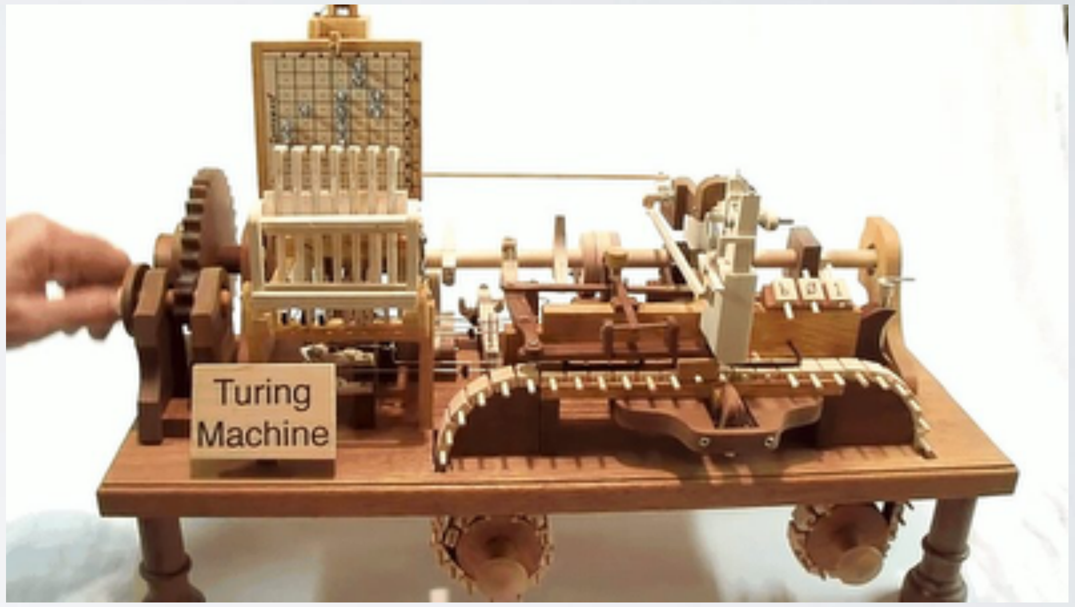
```
#include <stdio.h>
void main (void) {
  printf(“Hello, world.\n”);
}
```



Turing machine that prints “Hello, world.”

https://github.com/shinh/elvm

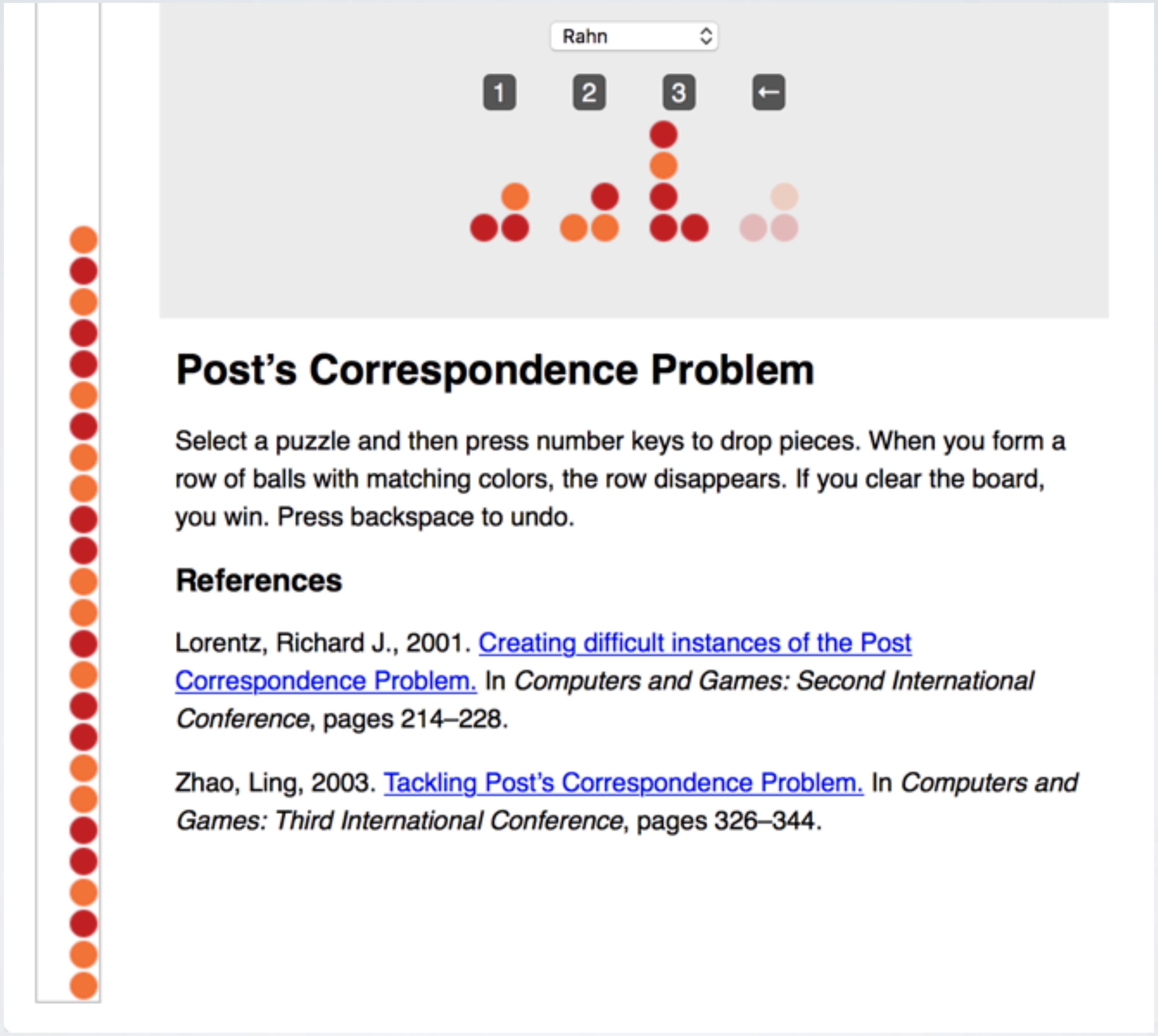
C program that simulates
Turing machines



Turing machine that simulates
Turing machines

https://github.com/shinh/elvm

But there is no Turing machine that can solve PCP



Post's Correspondence Problem

Select a puzzle and then press number keys to drop pieces. When you form a row of balls with matching colors, the row disappears. If you clear the board, you win. Press backspace to undo.

References

Lorentz, Richard J., 2001. Creating difficult instances of the Post Correspondence Problem. In *Computers and Games: Second International Conference*, pages 214–228.

Zhao, Ling, 2003. Tackling Post's Correspondence Problem. In *Computers and Games: Third International Conference*, pages 326–344.

Unit 1–2: Restricted machines

Unit 3: Computability and uncomputability

Unit 4: Tractability and intractability

```
1. if read a: write a; move right; goto 2
2. if read a: write a; move left; goto 1
3. halt
```

SECURITY

Emergency Windows Meltdown patch locks some AMD PCs into endless loop

After installing the update users say their PCs are unable to boot and eventually get stuck in an endless loop, as they try to roll back to an earlier version of the OS.

By Nick Heath | January 8, 2018, 3:48 AM PST

M. O. Rabin[*]

D. Scott[†]

Finite Automata and Their Decision Problems[‡]

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function

a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang[1] and by E. F. Moore,[4] and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape" machines seem to be new.

In Sections 1-6 the definition of the one-tape, one-way automaton is given and its theory fully developed. These machines are considered as "black boxes" having only a finite number of internal states and reacting to their environment in a deterministic fashion.

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications.

Many equivalent forms of the idea of finite automata have been published. One of the first of these was the definition of "nerve-nets" given by McCulloch and Pitts.[3] The theory of nerve-nets has been developed by authors too numerous to mention. We have been particularly influenced, however, by the work of S. C. Kleene[2] who proved an important theorem characterizing the possible action of such devices (this is the notion of "regular event" in Kleene's terminology). J. R. Myhill, in some unpublished work, has given a new treatment of Kleene's results and this has been the actual point of departure for the investigations presented in this report. We have not, however, adopted Myhill's use of directed graphs as a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang[1] and by E. F. Moore,[4] and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape" machines seem to be new.

In Sections 1-6 the definition of the one-tape, one-way automaton is given and its theory fully developed. These machines are considered as "black boxes" having only a finite number of internal states and reacting to their environment in a deterministic fashion.

We center our discussions around the application of automata as devices for defining sets of tapes by giving "yes" or "no" answers to individual tapes fed into them. To each automaton there corresponds the set of those tapes "accepted" by the automaton; such sets will be referred to as *definable sets*. The structure of these sets of tapes, the various operations which we can perform on these sets, and the relationships between automata and defined sets are the broad topics of this paper.

After defining and explaining the basic notions we give, continuing work by Nerode,[5] Myhill, and Shepherdson,[7] an intrinsic mathematical characterization of definable sets. This characterization turns out to be a useful tool for both proving that certain sets are definable by an automaton and for proving that certain other sets are not.

In Section 4 we discuss decision problems concerning automata. We consider the three problems of deciding whether an automaton accepts any tapes, whether it ac-

*Now at the Department of Mathematics, Hebrew University in Jerusalem.

†Now at the Department of Mathematics, University of Chicago.

‡The bulk of this work was done while the authors were associated with the IBM Research Center during the summer of 1957.

Unit 1–2: Restricted machines

Unit 3: Computability and uncomputability

Unit 4: Tractability and intractability

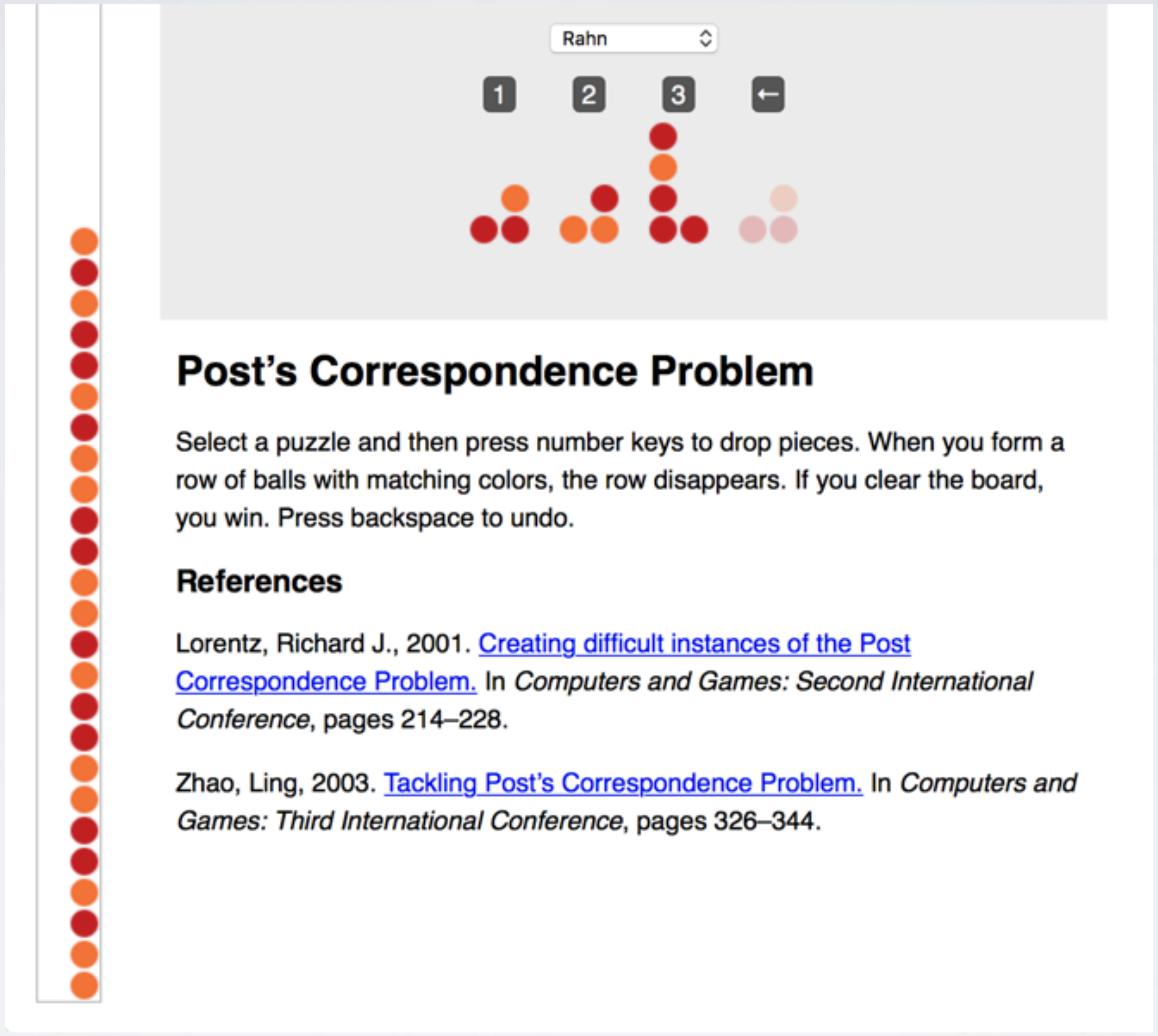
ON THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS

BY
J. HARTMANIS AND R. E. STEARNS

I. **Introduction.** In his celebrated paper [1], A. M. Turing investigated the computability of sequences (functions) by mechanical procedures and showed that the set of sequences can be partitioned into computable and noncomputable sequences. One finds, however, that some computable sequences are very easy to compute whereas other computable sequences seem to have an inherent complexity that makes them difficult to compute. In this paper, we investigate a scheme of classifying sequences according to how hard they are to compute. This scheme puts a rich structure on the computable sequences and a variety of theorems are established. Furthermore, this scheme can be generalized to classify numbers, functions, or recognition problems according to their computational complexity.

The computational complexity of a sequence is to be measured by how fast a multitape Turing machine can print out the terms of the sequence. This particular abstract model of a computing device is chosen because much of the work in this area is stimulated by the rapidly growing importance of computation through the use of digital computers, and all digital computers in a slightly idealized form belong to the class of multitape Turing machines. More specifically, if $T(n)$ is a computable, monotone increasing function of positive integers into positive integers and if α is a (binary) sequence, then we say that α is in complexity class

Bounded PCP: Use at most *n* pieces



Post's Correspondence Problem

Select a puzzle and then press number keys to drop pieces. When you form a row of balls with matching colors, the row disappears. If you clear the board, you win. Press backspace to undo.

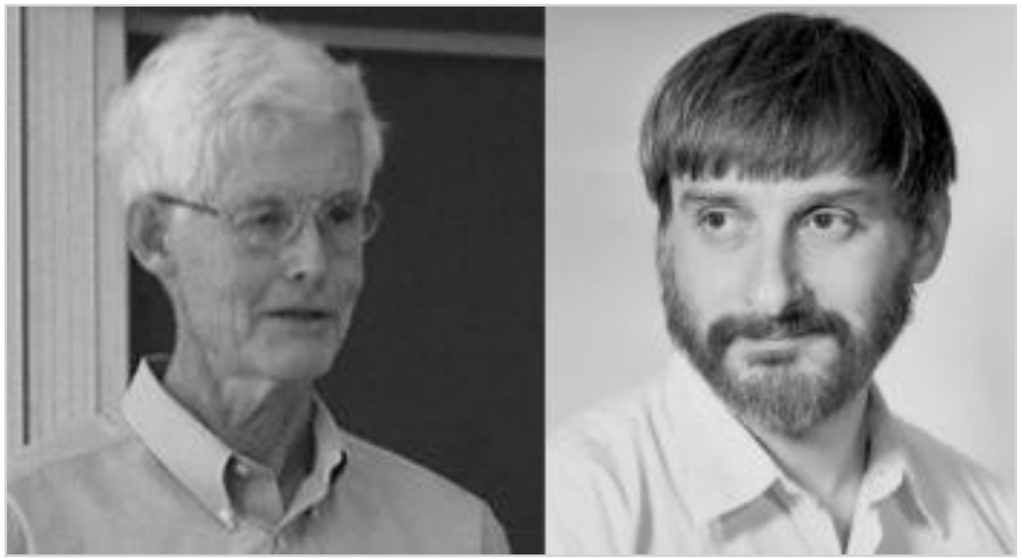
References

Lorentz, Richard J., 2001. Creating difficult instances of the Post Correspondence Problem. In *Computers and Games: Second International Conference*, pages 214–228.

Zhao, Ling, 2003. Tackling Post's Correspondence Problem. In *Computers and Games: Third International Conference*, pages 326–344.

Is there a polynomial-time ($O(n^k)$) solution?

P vs NP Problem



Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971

Rules:

Rules for the Millennium Prizes

Related Documents:

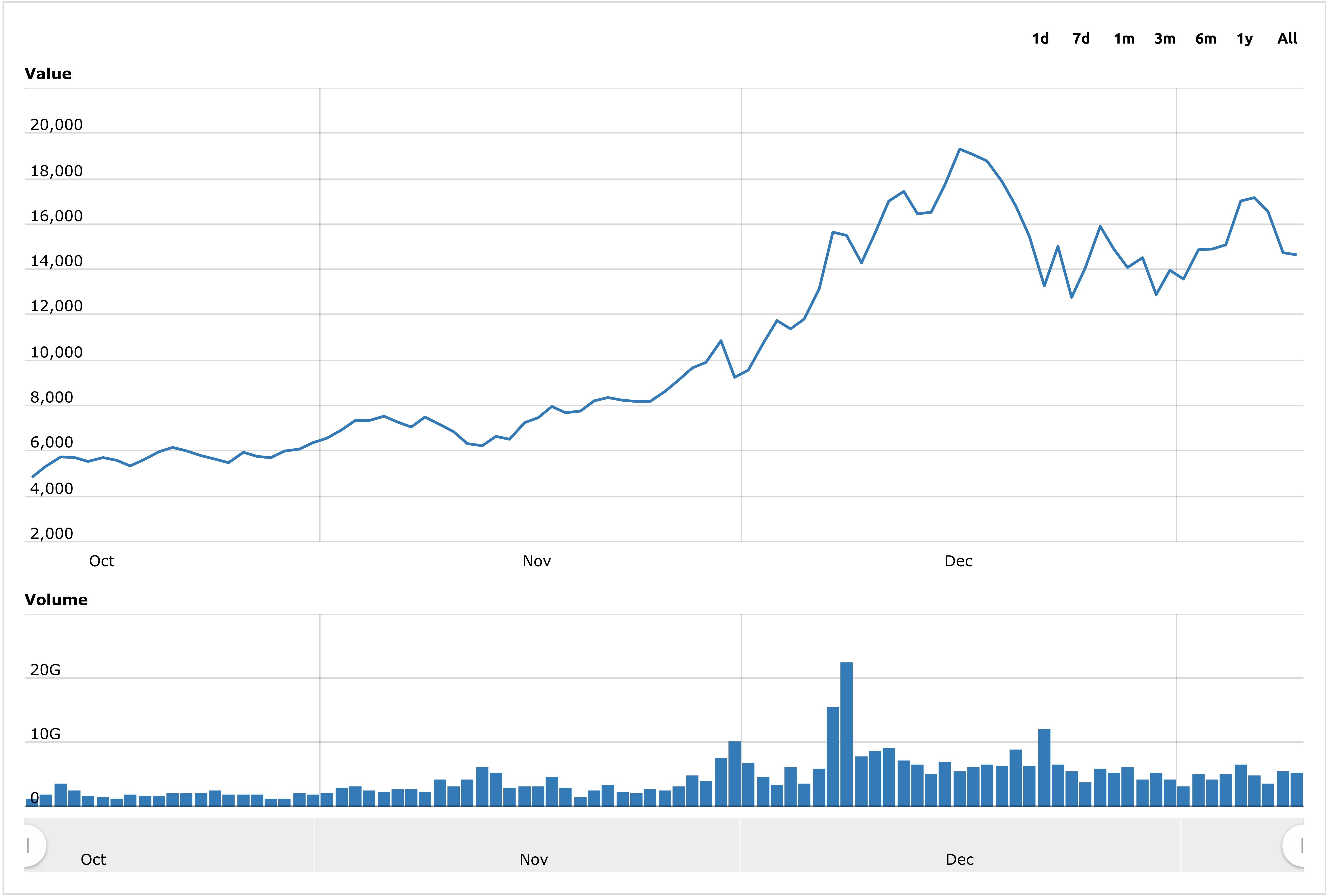
Official Problem Description

Minesweeper

Related Links:

Lecture by Vijaya Ramachandran

Bitcoin Charts



If such a procedure existed, then we could quickly find the smallest Boolean circuits that output (say) a table of historical stock market data, or the human genome, or the complete works of Shakespeare. It seems entirely conceivable that, by analyzing these circuits, we could make an easy fortune on Wall Street, or retrace evolution, or even generate Shakespeare’s 38th play. For broadly speaking, that which we can compress we can understand, and that which we can understand we can predict. Indeed, in a recent book, Eric Baum argues that much of what we call ‘insight’ or ‘intelligence’ simply means finding succinct representations for our sense data….So if we could solve the general case—if knowing something was tantamount to knowing the shortest efficient description of it—then we would be almost like gods. The NP Hardness Assumption is the belief that such power will be forever beyond our reach.

–Scott Aaronson

nd.edu/~dchiang/teaching/theory