Alan Turing, World War II, and the Theory of Computation

Ross Dempsey

Department of Physics and Astronomy
Johns Hopkins University

JHU Splash, 2018

Outline

1. Enigma in World War II
   - Enigma Machine
   - Polish Codebreaking Effort
   - Bletchley Cryptanalysis
   - Construction of Bombes
   - Consequences
2. Automata and Computing
   - Automata and Languages
   - Turing Machines
3. Theory of Computation
   - Undecidability
   - Complexity Classes

Invention of Enigma

- Enigma invented by Arthur Scherbius
- Implemented a more robust substitution cipher

$$A \to D$$
$$B \to A$$
$$C \to M$$
$$D \to T$$
$$\vdots$$

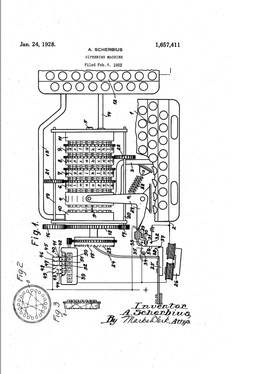


Figure: U.S. Patent 1,657,411 for the Enigma machine.

Adoption

- Adopted first by the *Reichsmarine* in 1926
  - Three rotors from a set of five
  - Reflector for extra encryption
- Over 100 billion configurations from the plugboard alone
- Gelmans considered Enigma **unbreakable**

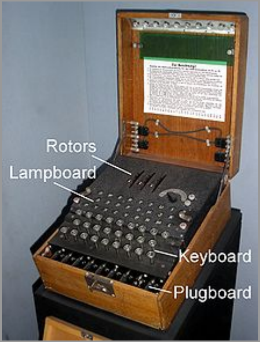


Figure: One of many military-enhanced Enigma designs, equipped with extra rotors and a plugboard.

Polish Cipher Bureau

- The Polish Cipher Bureau found an Enigma machine
- Marian Rejewski applied pure mathematics to its design
  - Conjugacy classes of pelmutations are given by cycle structure
- Decrypted 75% of messages before Gelmans changed scheme

Figure: Marian Rejewski, a Polish mathematician who made considerable progress in Enigma cryptanalysis.

Polish Contribution

- Five weeks before war, the Poles infolmed the Allies
- "Ultra would never have gotten off the ground if we had not learned from the Poles, in the nick of time, the details both of the Gelman military...Enigma machine, and of the operating procedures that were in use." – Gordon Welchman

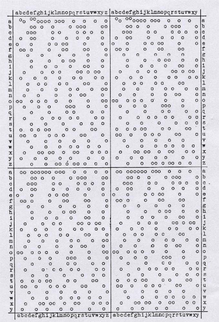


Fig. 5. Diagram of the perforated sheet

Figure: A perforared Zygalski sheet, one of the tools used by the Polish cryptanalysts.

Bletchley Park

- British Government Code & Cipher School (GC&CS) bought Bletchley in 1938
- Situated conveniently between Oxford and Cambridge
- Began recruiting "men of the professor type" before the war
    - Alan Turing
    - Gordon Welchman
    - Peter Twinn

Figure: Codebreakers arrive at the Bletchley Park mansion in 1939.

Crib Detection

- Decryption relied on "cribs," or known plaintext
- Cribs were based on weather reports and other predictable messages

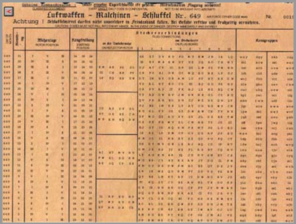


Figure: The British used cribs to detelmine the daily keys on Gelman keysheets.

Enigma Captures

- The Royal Navy assisted Bletchley by capturing Enigma equipment
  - Rotor wheels from U-33 in 1940
  - Keysheet from U-110 in 1941
- Sometimes cribs were planted and “gardened” by placing mines in known locations

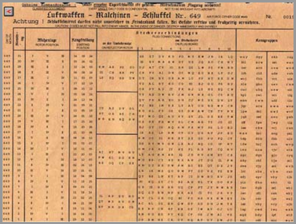

Luftwaffen - Maschinen - Schlüssel Nr. 649
Achtung !

Figure: The British used cribs to detelmine the daily keys on Gelman keysheets.

British Bombe

- To automate much of the cryptanalysis process, Alan Turing designed the *bombe*
- Starting in 1941, Wrens began operating bombes

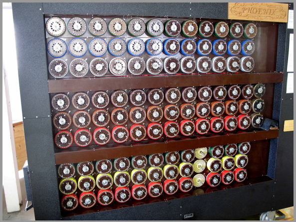


Figure: A working rebuilt bombe at the Bletchley Park museum.

British Bombe

http://www.youtube.com/v/Hb44bGY2KdU?rel=0

Battle of the Atlantic

- When GC&CS began decrypting messages at full capacity, shipping losses dropped by $2/3$
- Blackout in 1942 $\rightarrow$ increase in shipping losses $\rightarrow$ Alan Turing breaks TRITON
- Ten day blackout in 1943 $\rightarrow$ Britain near defeat $\rightarrow$ "Black May"

Figure: A British ship sinks a Gelman U-boat.

Overall Effects

"And altogether therefore the war would have been something like two years longer, perhaps three years longer, possibly four years longer than it was. . . . I think we would have won but it would have been a long and much more brutal and destructive war." – Harry Hinsley

Figure: American Liberty ships creating a sheltered area around Omaha beach.

Turing's Machine

https://www.youtube.com/v/M47hsaYWZE?rel=0

Automata

- How does a machine go about computing?
- Automata provide a *model of computation* which is well-suited for machines
- Different automata have different levels of computational ability
- Level of ability detelmined by which languages can be decided

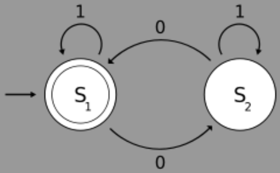


Figure: An example of a detelministic finite automaton

Detelministic Finite Automata

- A DFA is made of a finite number of states (circles)
- Each bit of input moves the machine to its next state (arrows)
- String is accepted if the machine finishes in accept state (double circle)
- Which of these strings are accepted?
  - 010101
  - 110011
  - 101001
- What set of strings, or *language*, is accepted?

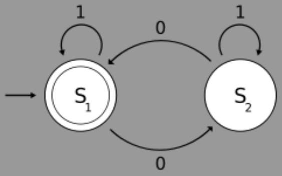


Figure: An example of a detelministic finite automaton

Regular Languages

- A regular language is encoded by a regular expression
- The essential ingredients for regular expressions are:
  - Kleene closure (*): $x^* = \{\varnothing, x, xx, xxx, \ldots\}$
  - Union ($|$): $x|y = \{x, y\}$
  - Concatenation: $xy = \{xy\}$
- What is the language $(1^*01^*01^*)^*$?

Divisibility by 5

- Binary strings are numbers; e.g., $10011_2 = 19_10$
- We will read numbers in reverse order: $11001 \to 19$
- Is it possible to construct a DFA to check for divisibility by 5?

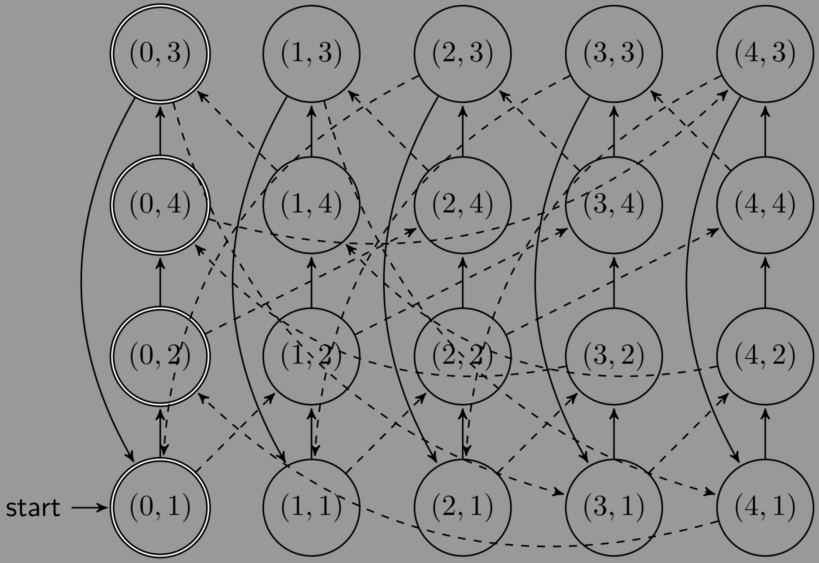
Divisibility by 5

- Binary strings are numbers; e.g., $10011_2 = 19_10$
- We will read numbers in reverse order: $11001 \rightarrow 19$
- Is it possible to construct a DFA to check for divisibility by 5?
- **Yes.** How many states will we need?

Divisibility by 5

- Binary strings are numbers; e.g., $10011_2 = 19_10$
- We will read numbers in reverse order: $11001 \to 19$
- Is it possible to construct a DFA to check for divisibility by 5?
- **Yes.** How many states will we need?
- **20**. Each state stores a remainder modulo 5 (0-4), and the remainder of the next power of 2 (1-4)

Sometimes Things Get Messy



DFAs decide Regular Languages

- There is an algorithm to convert between DFAs and regular expressions
- Every DFA accepts a regular language; every regular language is accepted by a DFA
- Which languages are regular?
    - Numbers divisible by n?
    - Strings with five 1s?
    - Strings with 2^n symbols?
    - Balanced strings $[0 \to (, 1 \to)]$?

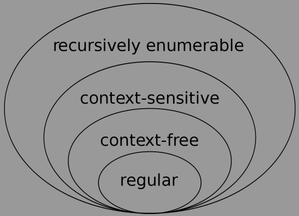


Figure: Regular languages do not include all interesting languages.

Decidable Languages

- The most interesting class of languages is all languages which can be decided by any finite process
- What automata can decide these languages?

Figure: A slate statute of Alan Turing holding the Enigma.

Decidable Languages

- The most interesting class of languages is all languages which can be decided by any finite process
- What automata can decide these languages?
- Turing machines can decide all "recursively enumerable" languages
- It is believed that these are all the decidable languages

Figure: A slate statute of Alan Turing holding the Enigma.

Turing Machines

- A Turing machine is similar to a DFA: it has a collection of states, with rules for moving between them
- The input is given on a *Turing tape*
- Turing machines can move along their tape and modify it one step at a time

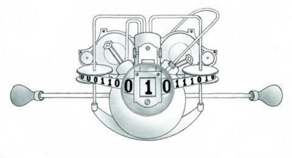


Figure: Turing machines use a tape to read and write memory.

Example: Strings of Length 2^n

- We determined that strings of length 2^n do not form a regular language
- Does the addition of a Turing tape allow us to decide this language?

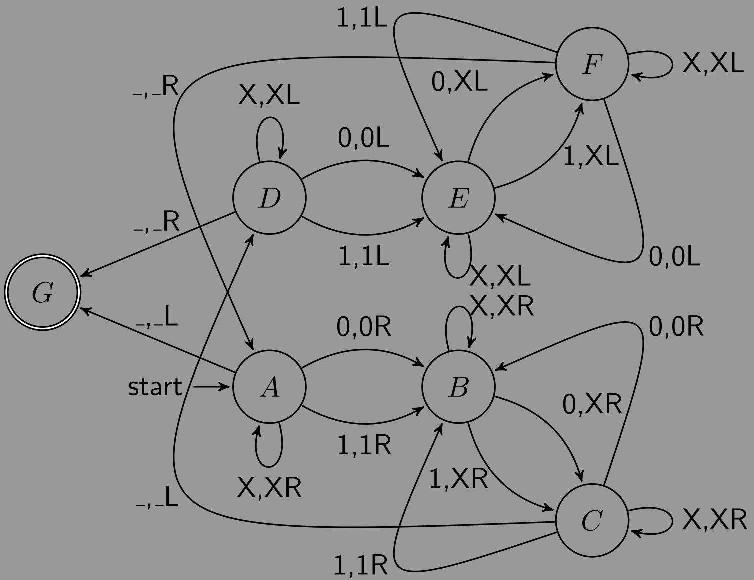
Example: Strings of Length 2^n

- We determined that strings of length 2^n do not form a regular language
- Does the addition of a Turing tape allow us to decide this language?
- Idea: repeatedly halve length of string.
  $1001 \to 1X0X \to XX0X \to XXXX$

Example: Strings of Length 2^n

- We determined that strings of length 2^n do not form a regular language
- Does the addition of a Turing tape allow us to decide this language?
- Idea: repeatedly halve length of string.
  $1001 \to 1X0X \to XX0X \to XXXX$
- If not a power of two, the machine will notice:
  $110110 \to 1X0X1X \to$ reject

More Volunteers Please



Universal Turing Machine

- The diagram on the previous slide is akin to a computer program
- Computers can run arbitrary programs; can a Turing machine?
- Universal Turing Machine (UTM)
  - Takes another Turing machine and data as input
  - Simulates given Turing machine on the input
  - Can compute anything your laptop can compute (and more: the UTM has infinite memory)

Undecidable Languages

- Deterministic finite automata were limited to regular languages
- Turing machines are limited to decidable languages
  - How could a language be undecidable?

Undecidable Languages

- Deterministic finite automata were limited to regular languages
- Turing machines are limited to decidable languages
  - How could a language be undecidable?
  - Consider HALT, the language of inputs to a UTM which eventually terminate

Undecidable Languages

- Deterministic finite automata were limited to regular languages
- Turing machines are limited to decidable languages
  - How could a language be undecidable?
  - Consider HALT, the language of inputs to a UTM which eventually terminate
  - What if there were a Turing machine T which decided HALT?

Halting Problem is Undecidable

- Proof: let T be a Turing machine which accepts strings in HALT

Halting Problem is Undecidable

- Proof: let T be a Turing machine which accepts strings in HALT
- Let T' be a Turing machine which simulates T on its input and:
  - If T accepts, T' enters an infinite loop
  - If T rejects, T' terminates

Halting Problem is Undecidable

- Proof: let T be a Turing machine which accepts strings in HALT
- Let T' be a Turing machine which simulates T on its input and:
  - If T accepts, T' enters an infinite loop
  - If T rejects, T' terminates
- Now run the machine T' on the input T'
  - If T' halts, then T accepts, and then T' enters an infinite loop $\rightarrow$ contradiction
  - If T' enters an infinite loop, then T rejects, and T' terminates $\rightarrow$ contradiction

Decidable Languages

- Not all decidable languages – i.e., solvable problems – are equal
- Consider the following languages:
  - $\mathrm{SORT} = \{$sorted lists of integers$\}$
  - $\mathrm{LIN\text{-}SOLVABLE} =$ $\{$linear systems of equations which have solutions$\}$
  - $\mathrm{SAT} = \{$Boolean expressions that are satisfiable$\}$

Decidable Languages

- Not all decidable languages – i.e., solvable problems – are equal
- Consider the following languages:
  - $\mathrm{SORT} = \{$sorted lists of integers$\}$
  - $\mathrm{LIN\text{-}SOLVABLE} =$ $\{$linear systems of equations which have solutions$\}$
  - $\mathrm{SAT} = \{$Boolean expressions that are satisfiable$\}$
- SORT can be decided in *linear* time, by reading through the list

Decidable Languages

- Not all decidable languages – i.e., solvable problems – are equal
- Consider the following languages:
  - $\mathrm{SORT} = \{$sorted lists of integers$\}$
  - $\mathrm{LIN\text{-}SOLVABLE} =$ $\{$linear systems of equations which have solutions$\}$
  - $\mathrm{SAT} = \{$Boolean expressions that are satisfiable$\}$
- SORT can be decided in *linear* time, by reading through the list
- $\mathrm{LIN\text{-}SOLVABLE}$ can be decided in *cubic* time, using Gaussian elimination

Decidable Languages

- Not all decidable languages – i.e., solvable problems – are equal
- Consider the following languages:
  - $\mathrm{SORT} = \{$sorted lists of integers$\}$
  - $\mathrm{LIN\text{-}SOLVABLE} =$ $\{$linear systems of equations which have solutions$\}$
  - $\mathrm{SAT} = \{$Boolean expressions that are satisfiable$\}$
- SORT can be decided in *linear* time, by reading through the list
- $\mathrm{LIN\text{-}SOLVABLE}$ can be decided in *cubic* time, using Gaussian elimination
- SAT can be decided in *exponential* time, by trying all combinations of variables

Polynomial Time

- SORT and LIN-SOLVABLE can both be decided in a time which is a polynomial function of the input size
- This defines the complexity class **P**
- Problems in **P** are generally *tractable* – polynomials grow slowly enough that large input sizes are OK

Nondeterministic Polynomial Time

- SAT (probably) can't be solved in polynomial time
- But a particular assignment of variables can be checked in linear (polynomial) time
- If we had a *nondeterministic* Turing machine, which could check all possibilities at once, it could solve SAT in polynomial time
- We say $\mathrm{SAT} \in$ **NP**, the class of nondeterministic polynomial time problems

P vs. **NP**

- Many problems are known to be in **P**, and many are known to be in **NP**
- It is known that $\mathbf{P} \subseteq \mathbf{NP}$ – how would you prove this?
- It is strongly believed but *not* known that $\mathbf{P} \subsetneq \mathbf{NP}$
- This is the famous **P** vs. **NP** problem