Post Correspondence Problem

Slides: https://www.andrew.cmu.edu/user/ko/pdfs/lecture-17.pdf

Suppose we have dominos

$$\left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

A match is a list of these dominos so that when concatenated the top and the bottom strings are identical. For example,

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right] = \frac{abcaaabc}{abcaaabc}$$

Post Correspondence Problem

An instance of the PCP

A PCP instance over Σ is a finite collection P of dominos

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \dots, \left[\frac{t_k}{b_k}\right] \right\}$$

where for all i, $1 \leq i \leq k$, $t_i, b_i \in \Sigma^*$.

Match

Given a PCP instance P, a match is a nonempty sequence

$$i_1, i_2, \dots, i_\ell$$

of numbers from $\{1, 2, \dots, k\}$ (with repetition) such that
$t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}$

Post Correspondence Problem

QUESTION:

Does a given PCP instance P have a match?

LANGUAGE FORMULATION:

$PCP = \{\langle P \rangle \mid P \text{ is a PCP instance and it has a match}\}$

THEOREM 5.15

PCP is undecidable.

Proof: By reduction using computation histories. If PCP is decidable then so is A_{TM}. That is, if PCP has a match, then M accepts w.

PCP – Adding the right kind of dominos

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right],$$

PCP – Adding the right kind of dominos

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q, a) = (r, b, R), \text{ put } \left[\frac{qa}{br}\right] \text{into } P'$$

PCP – Adding the right kind of dominos

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q, a) = (r, b, R), \text{ put } \left[\frac{qa}{br}\right] \text{into } P'$$

3. Handle left moving transitions. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q, a) = (r, b, L), \text{ put } \left[\frac{cqa}{rcb}\right] \text{into } P'$$

PCP – Adding the right kind of dominos

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q, a) = (r, b, R), \text{ put } \left[\frac{qa}{br}\right] \text{into } P'$$

3. Handle left moving transitions. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q, a) = (r, b, L), \text{ put } \left[\frac{cqa}{rcb}\right] \text{into } P'$$

4. For every $a \in \Gamma$ put $\left[\frac{\mathtt{a}}{\mathtt{a}}\right]$ into P'

PCP – Adding the right kind of dominos

1. The first domino kicks of the computation history

$$\left[\frac{t_1}{b_1}\right] = \left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right],$$

2. Handle right moving transitions. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q,a) = (r,b,R), \text{ put } \left[\frac{qa}{br}\right] \text{into } P'$$

3. Handle left moving transitions. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{reject}$

$$\text{if } \delta(q,a) = (r,b,L), \text{ put } \left[\frac{cqa}{rcb}\right] \text{into } P'$$

4. For every $a \in \Gamma$ put $\left[\frac{\mathbf{a}}{\mathbf{a}}\right]$ into P'
5. Put $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$ into P'.

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\left[\frac{\#}{\#\ q_0\ 0\ 1\ 0\ 0\ \#}\right]$$

PCP - HOW THE DOMINOS WORK

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccc} \# & q_0 & 0 & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{cccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$
\begin{array}{ccccccccccccc}
\# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\
\# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \#
\end{array}
$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \# \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$
- What exactly is going on ?

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$
\begin{array}{ccccccccccccc}
\# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\
\# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \#
\end{array}
$$

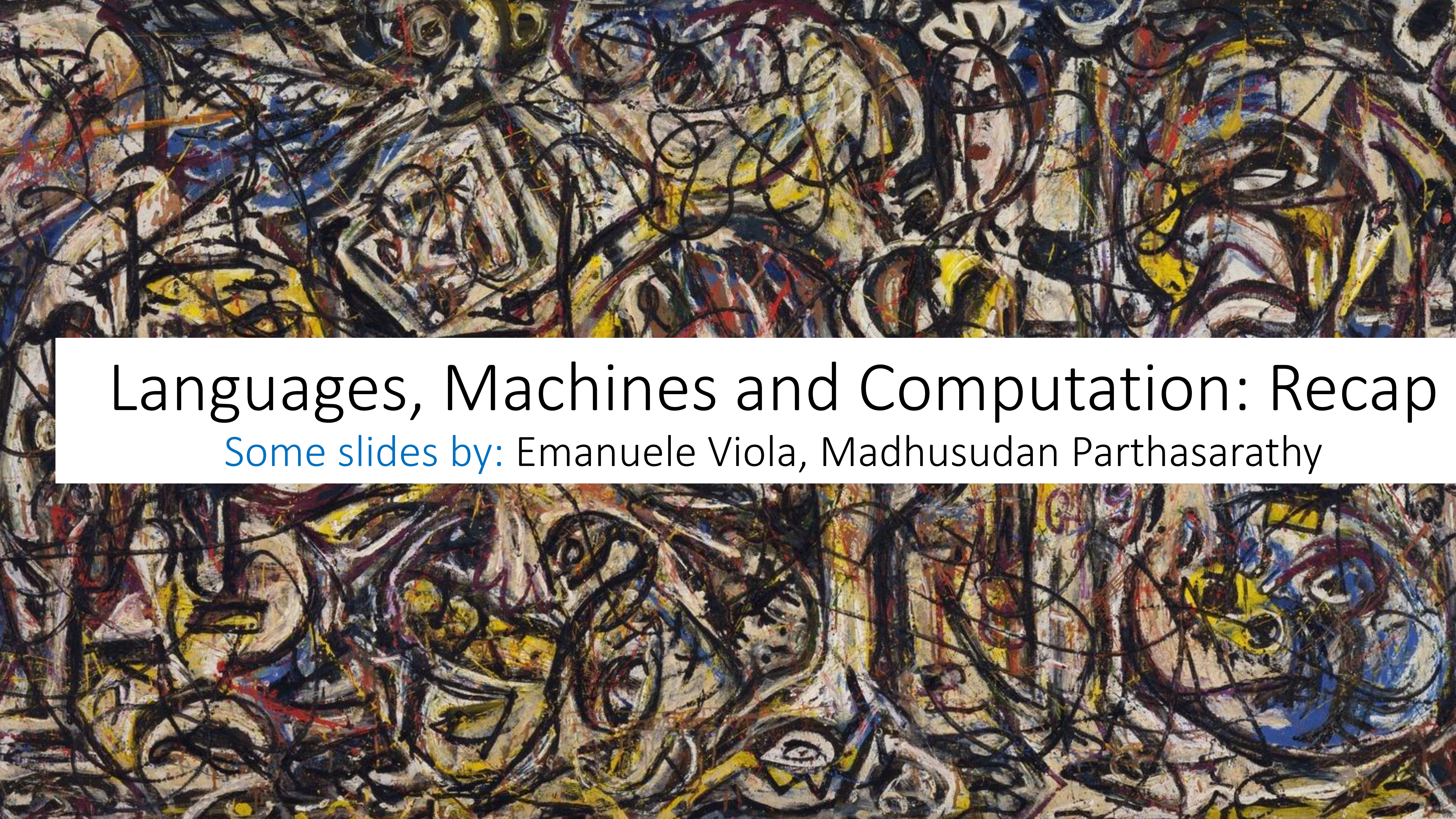
- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$
- What exactly is going on ?
- We force the bottom string to create a copy on the top which is forced to generate the next configuration on the bottom – We are simulating M on w!

PCP - How the dominos work

- Let us assume $\Gamma = \{0, 1, 2, \sqcup\}$, $w = 0100$ and that $\delta(q_0, 0) = (q_7, 2, R)$
- Part 1 places the first domino and the match begins

$$\begin{array}{ccccccccccccc} \# & q_0 & 0 & 1 & 0 & 0 & \# & & & & & & \\ \# & q_0 & 0 & 1 & 0 & 0 & \# & 2 & q_7 & 1 & 0 & 0 & \# \end{array}$$

- Part 2 places the domino $\left[\frac{q_0 0}{2 q_7}\right]$
- Part 4 places the dominos $\left[\frac{0}{0}\right]$ $\left[\frac{1}{1}\right]$ $\left[\frac{2}{2}\right]$ and $\left[\frac{\sqcup}{\sqcup}\right]$ into P' so we can extend the match.
- Part 5 puts in the domino $\left[\frac{\#}{\#}\right]$
- What exactly is going on ?
- We force the bottom string to create a copy on the top which is forced to generate the next configuration on the bottom – We are simulating M on w!
- The process continues until M reaches a halting state and we then pad the upper string.



Languages, Machines and Computation: Recap

Some slides by: Emanuele Viola, Madhusudan Parthasarathy

What did we learn?

- Mathematical maturity
  - Key to success in a scientific career
  - Exposure to proofs and rigorous reasoning

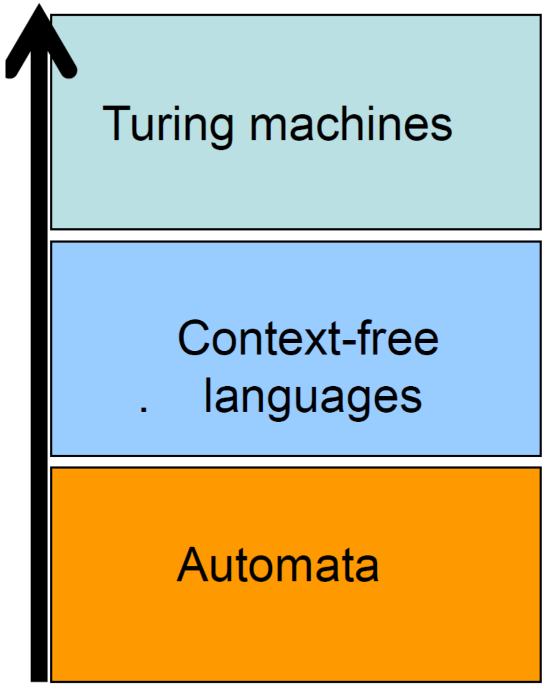
- Theory of computation
  - Develop models of computation and ask what can and cannot be computed by these models?
  - How quickly? With how much memory?

- Most famous question in CS, is P = NP? Millennium problem, $ 1 million prize

What did we learn?

- Understand the notion of computability
  - Define computation independent of physical computer
- Inherent limits of computability
- Tractability of weaker models of computation
- Relation of computability to formal languages

Models of Computation



- Finite automata: Computers with no memory
- Context-free grammars: Memory = stack.
- Turing machine: real computers, no bounds on memory

Key Classes of Languages

- Regular languages
  - Languages decided by finite-state machines
  - Robust, tractable
- Context-free languages
  - Languages expressed by CFGs
  - Decidable by machines
  - Semi-robust, semi-tractable
- Decidable Languages
  - The class of languages decidable using algorithms
  - Turing machine computable
  - Robust, not tractable

Finite Automata: Applications

- Finite automata: Computers with no memory
  - Examples: vending machines, switches etc
  - Lexical analysis in compilers
  - Searching for patterns: unix grep, web search, antivirus software
- Finite automata model protocols, electronic circuits.
  - Theory is used in *model-checking*.

Regular Languages

- DFAs = NFAs = RegExp
- Closed under union, intersection, complement, concatenation, *, reversal, …
- RegExp-> NFAs, NFAs -> RegExp, NFAs -> DFAs (subset construction; 2^n blowup)
- Suffix languages and Myhill-Nerode theorem:
  - L is regular iff L has finitely many suffix languages (equivalence classes)
  - Hence minimal DFAs exist (one state for every suffix language)
  - Efficient minimization of DFAs.
- Pumping Lemma
- Decidable problems: acceptance, equality, emptiness

Context Free Languages

- CFG = PDA
- Closed under union, concatenation, reversal, …
- Not closed under intersection, complement
- Membership problem is decidable: CYK algorithm
- Decidable problems: acceptance, emptiness
- Undecidable problems: fullness, equality
- Non-CFL: pumping lemma

Context Free Languages: Applications

- Parsing
  - Natural languages (semantic web; understanding speech, understanding text)
  - Programming languages (compilers)
- Recursive automata/PDAs
  - Modelling software control
  - Recursive procedures give recursive automata models
  - Static analysis of software done using these models
  - Compilers use them to check safety (types) and to do optimizations.
- XML
  - XML is basically bracketed text encoding hierarchical data
  - <car> <make> Honda </make> <year> 2002 </year> </car>
  - Data-type definitions –CFGs expressing valid XML documents
  - Conformance checking to DTDs, etc. are solvable.

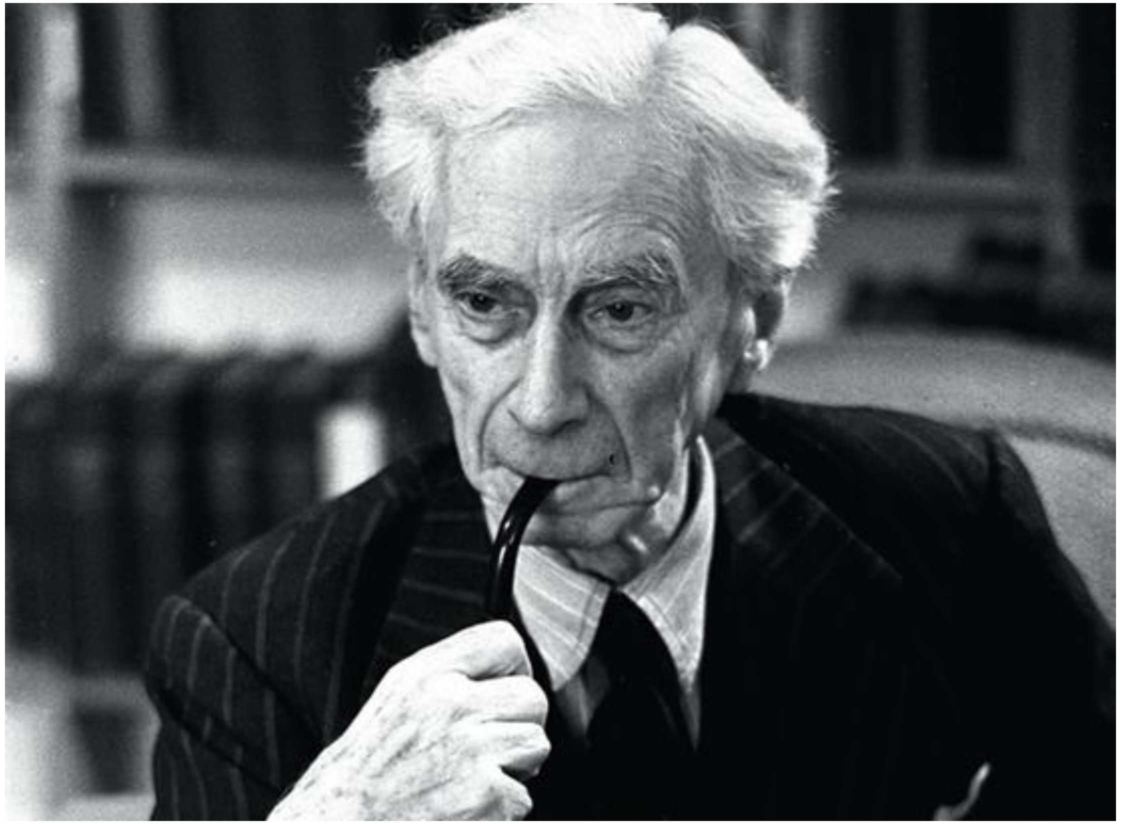
Decidable Languages

- Turing machines that halt
- Captures the class of problems solvable using "algorithms"
- Robust simple mathematical notion
  - independent of current knowledge of physics/engg
  - captures computability without using current proglang
- Closure under union, intersection, complement, concatenation, Kleene-*, reversal
- Nothing about the *language of a TM* is decidable (Rice's thm)
- Undecidable: Halting, acceptance, equality, emptiness…

More on Turing Machines

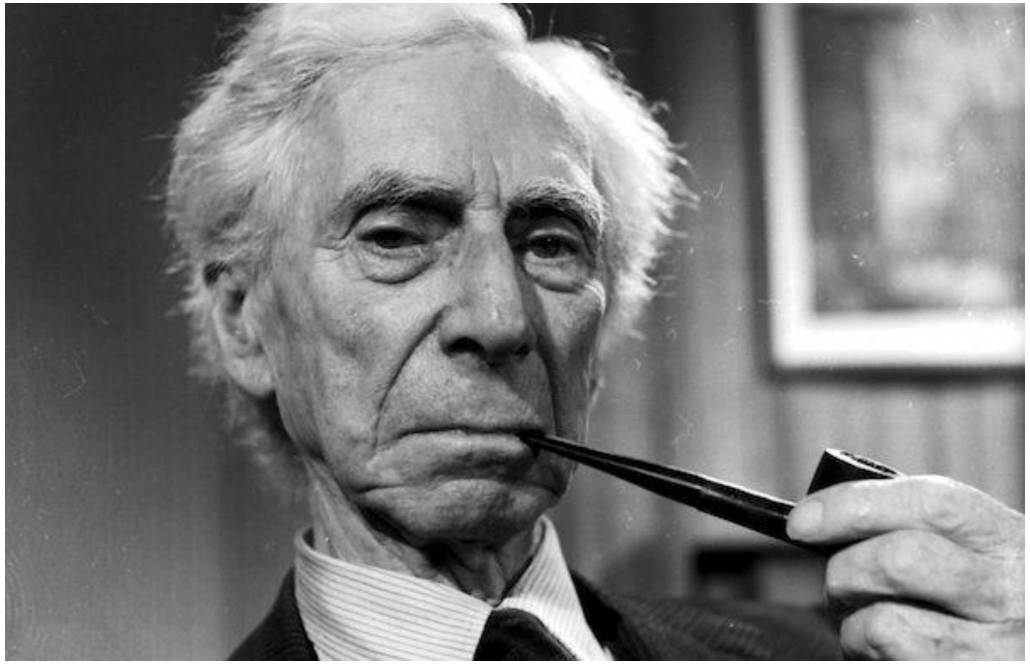
- Halting problem is undecidable (used diagonalization)
- Reductions: Reduce A to B so that solution to B gives solution to A
- If A reduces to B and B is decidable, then A is decidable.
- If A reduces to B and A is undecidable, then B is undecidable.
- Reductions: Direct, via computation histories
- Simple undecidable problem:
  - Post Correspondence Problem
  - Given a set of polynomial equations is there an integer valuation of the variables that satisfies the equations?

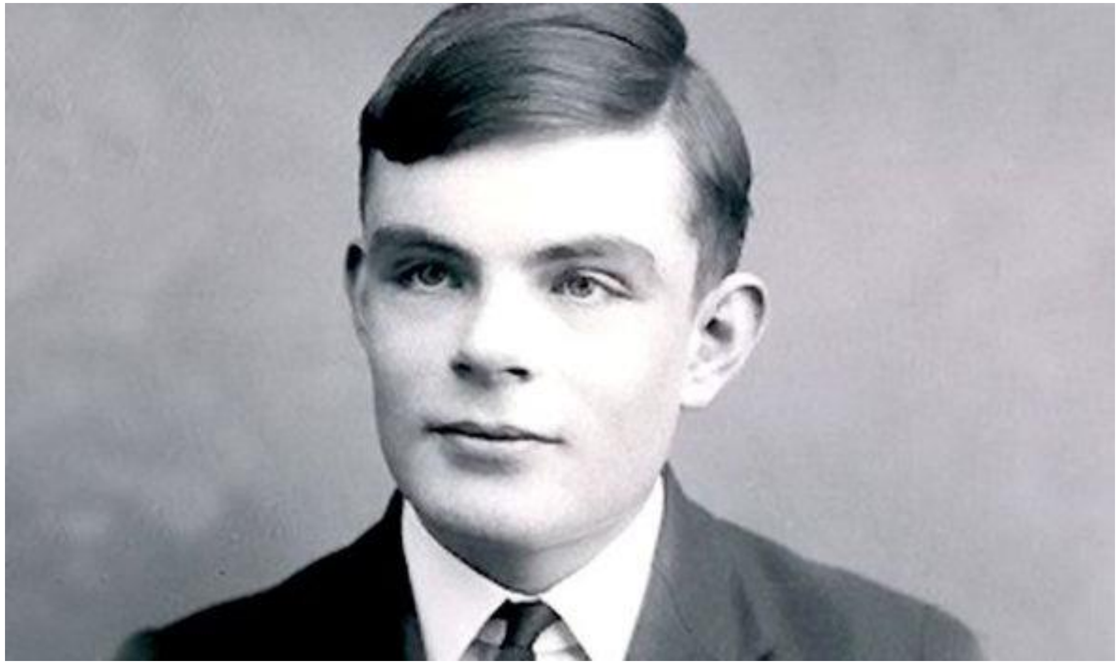
Russell’s paradox: Let us call a set "abnormal" if it is a member of itself, and "normal" otherwise. Now we consider the set of all normal sets, *R*.



Is R normal or abnormal?

What is Turing’s answer to Russell?





Learn to love abstraction.

Hope you enjoyed the course!