Lecture 6
Intro to Theory of Computation

So far...

\[ \text{2-tape } \text{TM} \equiv \text{1-tape TL} \]
\[ \text{NTM} \equiv \text{TM} \]
\[ \text{2-dim TM} \equiv \text{TM} \text{ (Homework)} \]

Also, \( \text{TM} \equiv \text{Random Access Machine} \)

Random Access Machine

- Just like model for ordinary assembly language
- except that
  - each register holds an arbitrary integer \( (\text{initially}) \)
  - there is one register for each natural number \( 0, 1, 2, \ldots \)
  - indirect addressing still allowed.
  - register indexed uses absolute value

Simulation: Configuration - store value of each register touched between \# - \# on tape

\( \text{eg. } \# \# 1 \# 2 \# \# \) (rest are implicitly 0)

Thus \( C, \text{Java, any programming language} \equiv \text{TM} \)

Also \( \text{TM} \equiv \lambda \text{-calculus} \equiv \mu \text{-recursive functions} \)

\( \text{Turing 1936} \quad \text{Kleene 1937} \)
Church–Turing Thesis (1936)

Any reasonable model that captures all of computation is equivalent in power to Turing machine.

Not a statement that can be proved or disproved since it uses a notion "reasonable model" which is not formal.

The text phrases this as:

\[ \text{Algorithm} \equiv \text{TM Algorithm} \]

But this only makes sense if the left side is meant to say "our intuitive notion of algorithm."

Since it was put forth, many other models of computation have been developed and all models are either

- equivalent to TM in power (or weaker)
- allow unreasonable operations within infinite action in a single step

Such models sometimes called "Turing-complete"
From now on we use high-level descriptions just as in ordinary pseudocode, not worrying about direction of head movements etc.

**Input encoding:**
Many problems have multiple input encodings.

**Example:** Graphs $G = (V, E)$
- Adjacency matrix
- Adjacency lists (e.g., singly, doubly linked)
- Edge lists

A TM can convert any one of these encodings to the other so we don't care which one.

**This is a string over some fixed alphabet**

We use $\langle G \rangle$ to denote any reasonable encoding of a graph $G$ in some fixed alphabet in \LaTeX{}:

\[
\{ \langle G \rangle \mid G \text{ is a graph} \} \subseteq \text{an input like any high level graph algorithm}
\]

More generally, we can encode many things between angle brackets.

**Example:** $\langle G, s, t \rangle$ : $G$ is a graph with a path from node $s$ to node $t$.3
on $\Sigma^* \times \Sigma^*$ encodes a pair of strings

eg. Input is a multivariate polynomial $p$

$$p = 6x^2y + 3y^2 + 3 \text{ in vars } x,y,z.$$

$\varphi$ : list of coefficients and monomials

Can also use $\langle M \rangle$ to denote encoding of computational device
eg. finite state machines

DFA, NFA, ...

labelled graph

eg. define $E_{\text{DFA}} = \{ \langle M \rangle : M \text{ is a DFA with } L(M) \neq \emptyset \}$

Recall a DFA $M$ has

$Q$, $\Sigma$, $\delta$, $q_0$, $F$,

- $Q$ : set of states
- $\Sigma$ : set of symbols
- $\delta$ : transition function
- $q_0$ : start state
- $F$ : set of accepting states

$\Sigma$ encoded is similar to that of graphs
must encode each symbol in $\Sigma$ since $\langle M \rangle$ is over a fixed alphabet


Thm. \text{EDFA is decidable}

Proof. For a DFA, \( L(M) \) is empty if no path from the start state \( q_0 \) to any of the states in \( F \).

Algorithm: Do a graph search (DFS, BFS, ...) in diagram of \( M \), starting at \( q_0 \).

- if no state in \( F \) reached, reject
- if state in \( F \) reached, accept

\( \text{ENFA} = \{ \langle M \rangle : M \text{ is an NFA and } L(M) = \emptyset \} \)

Thm. \text{ENFA is decidable}

Proof. Algorithm 1: Convert \( \langle M \rangle \) to \( \langle M' \rangle \) where \( M' \) is a DFA with \( L(M') = L(M) \).

Using subset construction.

- Run decider for EDFA on input \( \langle M' \rangle \).

Algorithm 2: (Same as algorithm for EDFA on diagram of \( M \)) check if state of \( F \) reachable from \( q_0 \).
ADFA = \{ <M, w> : M is a DFA that accepts string w \}

**Theorem:** ADFA is decidable

**Proof:** TM can do a step-by-step simulation of M on input w.

- Keep pointer on next char to be read (on tape)
- Record current state of M (on tape)
- Need this since M might have more states than the TM

Accept if state in F reached at end of w

ANFA = \{ <M, w> : NFA M accepts w \}

**Theorem:** ANFA is decidable

**Proof:** Algorithm 1: Convert input \( <M, w> \) to NFA \( M \) to \( <M', w> \) for equivalent DFA \( M' \).
- Run decision for ADFA on \( <M', w> \)
Algorithm 2: Do an "on-the-fly" simulation keeping track of all states reachable from go after reading each character of w.

\[
\text{AReg} = \varepsilon < R, w > : R \text{ is a regular expression that generates } w
\]

Then \text{AReg} is decidable

Proof: On input \(< R, w >\) convert \(R\) to an equivalent NFA \(M\) and run decision for

\[
\text{ANFA on input } < M, w >
\]

\[
\text{EQ}_{\text{DFA}} = \{ < M_1, M_2 > : M_1 \text{ and } M_2 \text{ are DFAs with } L(M_1) = L(M_2) \}
\]

Then \text{EQ}_{\text{DFA}} is decidable

Proof: We give two algorithms based on
1. closure properties of DFAs
2. minimization alg for DFA
Closure

Recall from 311

Given DFAs \( M, M' \) we can build new DFA for:
- Complement \( \overline{L(M)} \)
  - change \( F \) to \( Q \sim F \)
- \( L(M) \cup L(M') \)
- \( L(M) \cap L(M') \)
- Cross-product construction

\[
\begin{align*}
L(M_1) = L(M_2) & \iff o(1) = o(2) \\
o(1) = L(M_1) \cap \overline{L(M_2)} \\
o(2) = L(M_2) \cap \overline{L(M_1)}
\end{align*}
\]

Algorithm: Build DFA \( <M> \) set

\[
L(M) = (L(M_1) \cap L(M_2)) \cup (L(M_2) \cap \overline{L(M_1)})
\]

- Run decision for E DFA on input \( <M> \) and output its answer

Minimization:
Recall DFA minimization alg from 311.
We didn’t prove it but it turns out that if \( L(M_1) = L(M_2) \)
Then the minimized versions of $M_1$ and $M_2$ will be the same up to renaming of states (which is easy to check since edge labels need to match).

Algorithm 2: Run minimize $\mathcal{D}_9$ on $M_1$ and on $M_2$.

- Check that minimized forms are the same (up to state renaming).

Note: This second $\mathcal{D}_9$ is efficient in practice (much more than the first).

- In fact, this second algorithm was what was used to grade your CSE 311 finite states machine homework!