Lecture 5

CSE 431
Intro to Theory of Computation

Recall:

- \( L(M) = \{ w \in \Sigma^* \mid \text{there exists } u, v \in \Sigma^* \text{ s.t. } (u, w, v) \in \delta(M) \} \) is the language recognized by \( M \)

- \( M \) is decider iff \( M \) either accept or reject each string \( w \in \Sigma^* \)

- A language \( L \) is Turing-recognizable or recursively enumerable iff \( L = L(M) \) for some TM \( M \)

- A language \( L \) is decidable iff \( L = L(M) \) for some decider \( M \)

- Two machines are equivalent iff they have the same I/O behavior
  
  e.g., they have the same input alphabet and they accept/reject the same strings!

Last time

Theorem: For every \( k \)-tape TM \( M \), there is an equivalent 1-tape TM

Note: \( k \) was a fixed constant independent of the input. Need to know \( k \) to define the 1-tape TM.
Nondeterministic TMs (NtMs)

conceptual, not practical model

Recall DFA vs. NFA

\[
\begin{align*}
\text{DFA:} & \quad a \xrightarrow{\theta} \eta \\
\text{many possible moves given } p, a
\end{align*}
\]

\[
\begin{align*}
\text{NFA:} & \quad a \xrightarrow{\alpha} \not\exists \\
\text{one possible move given } p, a
\end{align*}
\]

We saw that NFA's were convenient but for every NFA there was an equivalent DFA (though it required exponentially more states in the worst case).

With NtMs, we get the same option

\[
\begin{align*}
\text{DFA:} & \quad a \xrightarrow{\alpha} \not\exists \\
\text{many possible moves given } p, a
\end{align*}
\]

\[
\begin{align*}
\text{NFA:} & \quad a \xrightarrow{\alpha} \not\exists \\
\text{one possible move given } p, a
\end{align*}
\]

Formally, only change is that now

\[
\delta : Q \times \Gamma \rightarrow \wp (Q \times \Gamma \times \Sigma \times R_{23})
\]

\[
\text{Down set}
\]

\[
\text{use a set of possible moves, not just one}
\]
Theorem: For every NTM there is an equivalent TM

Proof idea: Graph search: explore every node in computation tree, stopping early if acceptance is found

Computation tree

Graph Search: Optimize
  
  DFS x
  BFS ✓

Bad because might get stuck on infinite path

fan-out of every node

it is at most

\[ b = \lfloor \log_b \mathcal{L} \rfloor \]

which is the max # of possible next moves for a configuration.

Associate each move with a number from 0, ..., b-1

- Each node in tree at level \( t \) associated with a base \( b \) string of length \( t \)
- address of \( v \).

- Some addresses might not be reachable nodes

Overall idea: loop through all addresses (strings, base b) figuring out what the configuration would be at that node and stop & accept if that accepts
Implemented: 3 tapes

Repeat forever
  1. Copy input from input tape to work tape
  2. Use address on tape 2 to see which sequence of moves to try
     - Execute each move on work tape if legal for M
       - if not legal about this address
       - if queue reached then halt & accept
  3. Erase work tape
  4. Run machine to convert address on tape 2 to next address
     (just as with HW1 Problem 2 but for bigger b)

This will find a gap iff there is one.

Notes: If at some address length t all addresses above or reject, can reject.
Support original NTM accepted in \( T \) steps.

New TM will explore \( n \) \( b^T \) addresses (actually \( \sum_{f=0}^{F} b^T \)).

This is \( 2^{O(T)} \) since \( b \) is constant.

Thus \( \text{NP} \) question: can we reduce \( 2^{O(T)} \) to \( \text{poly}(T) \)?

---

**enumerator TM**

- no input
- read/write work tape initially blank
- write-only 1-way output tape
- alphabet: \( \Sigma = \{0, 1\} \)
- \( f = "\n" \)

Language enumerated by \( M \) is

\( E(M) \) the set of strings \( w \in \Sigma^* \) that \( M \) eventually prints (between two \( f \)'s on output tape)

Then \( L \) is Turing-recognizable if and only if \( L = E(M) \) is enumerable.
Proof \( (\Leftarrow) \) Suppose there is an enumerator 
\( M \) s.t. \( L = E(M) \)

2-tape TM \( M' \) recognizing \( L' \)
- on input \( w \), run \( M \) (with 2 other tapes)
  - Every time \( M \) print a 
    - Compare the string it just produced to \( w \). If they are equal accept.
  - If not just continue.

Clearly \( M' \) will accept \( w \) iff \( M \) prints \( w \).

\( (\Rightarrow) \) this is the harder direction.

Suppose we have a TM \( M'' \) with \( L = L(M'') \).
We need to build an enumerator for \( L \) that runs \( M'' \) on all possible strings in \( \Sigma^* \).

- We can generate all the strings in \( \Sigma^* \) one after another.

Using the counter method we used for addressed.

That's an infinite \( \# \) of strings, but just one computes may run forever!
Key Swiss idea: "Dovetailing"

Time step $t \leq N$

Strings

We need to try every string. For all possible $t$ at that step (find all the good)

 Enumerator $M'$:

For $t = 0, \ldots, \infty$ do

- For each of the first $t$ strings $w \in \Sigma^*$

  - Run $M'$ on input $w$ for $t$ steps
  - If $M'$ accepts print $w$

Eventually, will explore every point in above infinite table so will find and print all accepted strings

$	herefore E(M') = \mathbb{L}$