Lecture 12:

- Nondeterministic TMs
- Church-Turing Thesis
- Decidable Problems

Mark Bun
October 20, 2022
Nondeterministic TMs

At any point in computation, may nondeterministically branch. Accepts iff there exists an accepting branch.

Transition function \( \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, S\}) \)

- Transition function can lead to multiple states
- \( a \rightarrow b, R \)
- \( a \rightarrow b, R \)
- \( a \rightarrow b, R \)

- ...or give multiple write/movement instructions
- \( a \rightarrow b, R \)
- \( a \rightarrow c, L \)
- \( a \rightarrow c, L \)

- ...or both
- \( a \rightarrow b, R \)
- \( a \rightarrow c, L \)
Nondeterministic TMs

On input string w:
1) Scan tape left-to-right. At some point, nondeterministically go to step 2
2) a) Read the next symbol s and cross it off
   b) Move the head left repeatedly until a non-x symbol is found. If it matches s, cross it off. Else, reject.
   c) Move the head right until a non-x symbol is found. If blank is hit, go to step 3.
   d) Go back to 2a)
3) Check that the entire tape consists of x’s. If so, accept. Else, reject.

Language of this TM: \( \{ \omega \omega^R \} \mid \omega \in \{a,b,\lambda\}^* \)
Nondeterministic TMs

Ex. Given TMs $M_1$ and $M_2$, construct an NTM recognizing $L(M_1) \cup L(M_2)$

On input $w$,

1) Nondeterministically either:
   a) Run $M_1$ on $w$. Accept if it accepts.
   b) Run $M_2$ on $w$. Accept if it accepts.

If $w \in L(M_1) \cup L(M_2)$, then either $M_1$ or $M_2$ accepts $w$.
If $M_1$ accepts $w$, branch (a) leads $N$ to accept $w$.
If $M_2$ accepts $w$, branch (b) leads $N$ to accept $w$.
$\Rightarrow N$ accepts $w$.

If $w \notin L(M_1) \cup L(M_2)$, then no branch of computation leads $N$ to accept.
$\Rightarrow N$ does not accept $w$. 
Nondeterministic TMs

Ex. NTM for $L = \{w \mid w \text{ is a binary number representing the product of two integers } a, b \geq 2\}$

High-Level Description:

1) Nondeterministically guess $a \in \{2, \ldots, w^2\}$, $b \in \{2, \ldots, w^2\}$

2) Multiply $a \times b$, accept if $= w$.

Correctness analysis: (In our heads)

If $w \in L$, then $\exists a, b \in \{2, \ldots, w^2\}$ s.t. $a \times b = w$.

$\Rightarrow$ The branch of computation where $a, b$ guessed lead NTM to accept.

If $w \notin L$, no guess of $a, b$ will cause $a \times b = w$.

$\Rightarrow$ NTM does not accept.
Nondeterministic TMs

An NTM $N$ accepts input $w$ if when run on $w$ it accepts on at least one computational branch

$L(N) = \{w \mid N \text{ accepts input } w\}$

$N$ recognizes $L$ means:

- If $w \in L$, there exists a computational branch of $N$ on input $w$ that leads it to accept.
- If $w \notin L$, there exists a computational branch of $N$ on $w$ that loops forever.

An NTM $N$ is a decider if on every input, it halts on every computational branch

$N$ decides $L$ means:

- If $w \in L$, there exists an accepting branch.
- If $w \notin L$, every branch loops forever.
Non-deterministic TMs

**Theorem:** Every non-deterministic TM can be simulated by an equivalent deterministic TM

**Proof idea:** Explore “tree of possible computations”
Simulating NTMs

Which of the following algorithms is always appropriate for searching the tree of possible computations for an accepting configuration?

a) Depth-first search: Explore as far as possible down each branch before backtracking
   
   work if NTM is a decider

b) Breadth-first search: Explore all configurations at depth 1, then all configurations at depth 2, etc.
   
   Always works, even if NTM is a recognizer

c) Both algorithms will always work
Nondeterministic TMs

**Theorem:** Every nondeterministic TM has an equivalent deterministic TM

**Proof idea:** Simulate an NTM $N$ using a 3-tape TM
(See Sipser for full description)
TMs are equivalent to...

• TMs with “stay put”
• TMs with 2-way infinite tapes
• Multi-tape TMs
• Nondeterministic TMs
• Random access TMs
• Enumerators
• Finite automata with access to an unbounded queue
• Primitive recursive functions
• Cellular automata

...
Church-Turing Thesis

The equivalence of these models is a mathematical theorem (you can prove that each can simulate another)

Church-Turing Thesis v1: The basic TM (hence all of these models) captures our intuitive notion of algorithms

Church-Turing Thesis v2: Any physically realizable model of computation can be simulated by the basic TM

The Church-Turing Thesis is not a mathematical statement! Can’t be mathematically proved
Decidable Languages
1928 – The Entscheidungsproblem

The “Decision Problem”

Is there an algorithm which takes as input a formula (in first-order logic) and decides whether it’s logically valid?

Questions

- Can we automate the job of mathematicians?
- Can we automate the tasks we did in the first part of the course on regular languages?
Questions about regular languages

- Given a DFA $D$ and a string $w$, does $D$ accept input $w$?
- Given a DFA $D$, does $D$ recognize the empty language?
- Given DFAs $D_1, D_2$, do they recognize the same language?

(Same questions apply to NFAs, regexes)

**Goal:** Formulate each of these questions as a language, and decide them using Turing machines
Questions about regular languages

Design a TM which takes as input a DFA $D$ and a string $w$, and determines whether $D$ accepts $w$

How should the input to this TM be represented?

Let $D = (Q, \Sigma, \delta, q_0, F)$. List each component of the tuple separated by #

- Represent $Q$ by ,-separated binary strings
- Represent $\Sigma$ by ,-separated binary strings
- Represent $\delta : Q \times \Sigma \rightarrow Q$ by a ,-separated list of triples $(p, a, q), \ldots$

Denote the encoding of $D, w$ by $\langle D, w \rangle$
Representation independence

Computability (i.e., decidability and recognizability) is not affected by the precise choice of encoding.

\[ \text{Encodings } [\cdot], <\cdot> \]

\[ \exists \text{ a TM } M \text{ that, on input } [\cdot], \text{ outputs } <\cdot> \]

Why? A TM can always convert between different (reasonable) encodings.

Given a TM that recognizes \( L = \{ <0> \mid 0 \ldots \} \), construct a TM that recognizes \( L' = \{ [0] \mid 0 \ldots \} \).

On input \([0]\):

1) Run \( M([0]) \), producing \(<0>\).
2) Run recognizer for \( L \) on \(<0>\).

From now on, we’ll take \(<\_\>_\) to mean “any reasonable encoding”.
A “universal” algorithm for recognizing regular languages

\[ A_{\text{DFA}} = \{ \langle D, w \rangle \mid \text{DFA } D \text{ accepts } w \} \]

**Theorem:** \( A_{\text{DFA}} \) is decidable

**Proof:** Define a (high-level) 3-tape TM \( M \) on input \( \langle D, w \rangle \):

1. Check if \( \langle D, w \rangle \) is a valid encoding (reject if not)

2. Simulate \( D \) on \( w \), i.e.,
   - Tape 2: Maintain \( w \) and head location of \( D \)
   - Tape 3: Maintain state of \( D \), update according to \( \delta \)

3. **Accept** if \( D \) ends in an accept state, **reject** otherwise