Lecture 12:

- Nondeterministic TMs
- Church-Turing Thesis
- Decidable Problems

Reading:
Sipser Ch 3.2, 4.1

Mark Bun
October 20, 2022
Nondeterministic TMs

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

Transition function $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, S\})$
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.
Nondeterministic TMs

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

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

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

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

**Theorem:** Every nondeterministic 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

b) Breadth-first search: Explore all configurations at depth 1, then all configurations at depth 2, etc.

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)
TM\text{s are equivalent to...}

• TM\text{s with “stay put”}
• TM\text{s 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 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 $(p, a, q)$, ...

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

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

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

From now on, we’ll take $\langle \quad \rangle$ to mean “any reasonable encoding”
A “universal” algorithm for recognizing regular languages

\[ \mathcal{A}_{DFA} = \{ \langle D, w \rangle \mid \text{DFA } D \text{ accepts } w \} \]

Theorem: \(\mathcal{A}_{DFA}\) 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
Other decidable languages

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

$$A_{NFA} = \{ \langle N, w \rangle \mid \text{NFA } N \text{ accepts } w \}$$

$$A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates } w \}$$
NFA Acceptance

Which of the following describes a **decider** for $A_{\text{NFA}} = \{(N, w) \mid \text{NFA } N \text{ accepts } w\}$?

a) Using a deterministic TM, simulate $N$ on $w$, always making the first nondeterministic choice at each step. Accept if it accepts, and reject otherwise.

b) Using a deterministic TM, simulate all possible choices of $N$ on $w$ for 1 step of computation, 2 steps of computation, etc. Accept whenever some simulation accepts.

c) Use the subset construction to convert $N$ to an equivalent DFA $M$. Simulate $M$ on $w$, accept if it accepts, and reject otherwise.
Regular Languages are Decidable

**Theorem:** Every regular language $L$ is decidable

**Proof 1:** If $L$ is regular, it is recognized by a DFA $D$. Convert this DFA to a TM $M$. Then $M$ decides $L$.

**Proof 2:** If $L$ is regular, it is recognized by a DFA $D$. The following TM $M_D$ decides $L$.

On input $w$:

1. Run the decider for $A_{DFA}$ on input $\langle D, w \rangle$
2. Accept if the decider accepts; reject otherwise
Classes of Languages

- Regular
- Recognizable
- Decidable
More Decidable Languages: Emptiness Testing

Theorem: \( E_{DFA} = \{ \langle D \rangle \mid D \text{ is a DFA such that } L(D) = \emptyset \} \) is decidable.

Proof: The following TM decides \( E_{DFA} \)

On input \( \langle D \rangle \), where \( D \) is a DFA with \( k \) states:

1. Perform \( k \) steps of breadth-first search on state diagram of \( D \) to determine if an accept state is reachable from the start state.
2. Reject if a DFA accept state is reachable; accept otherwise.
$E_{DFA}$ Example

![DFA Example Diagram]
New Deciders from Old: Equality Testing

\[ EQ_{\text{DFA}} = \{ \langle D_1, D_2 \rangle \mid D_1, D_2 \text{ are DFAs and } L(D_1) = L(D_2) \} \]

Theorem: \( EQ_{\text{DFA}} \) is decidable

Proof: The following TM decides \( EQ_{\text{DFA}} \)

On input \( \langle D_1, D_2 \rangle \), where \( \langle D_1, D_2 \rangle \) are DFAs:

1. Construct DFA \( D \) recognizing the symmetric difference \( L(D_1) \Delta L(D_2) \)
2. Run the decider for \( E_{\text{DFA}} \) on \( \langle D \rangle \) and return its output
Symmetric Difference

$$A \triangle B = \{ w \mid w \in A \text{ or } w \in B \text{ but not both} \}$$