BU CS 332 – Theory of Computation

Lecture 12:
• More on NTMs
• Church-Turing Thesis
• Decidable Languages

Reading:
Sipser Ch 3.2, 4.1

Mark Bun
March 3, 2021
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

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

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

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

**NTM recognizers** can be simulated by deterministic TM recognizers

**NTM deciders** can be simulated by NTM deciders

$w \in L \Rightarrow$ there exists a branch leading to accept on input $w$

$w \notin L \Rightarrow$ all branches lead to reject
Nondeterministic TMs

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

On input $w$:
1. Nondeterministically guess $a, b \in \{2, ..., w\}$
2. Accept if $a \times b = w$, reject otherwise.

Proof of correctness:
If $w \in L$, there exist $\hat{a}, \hat{b}$ such that $\hat{a} \times \hat{b} = w$. Computation branch where we guessed $a = \hat{a}, b = \hat{b}$ accepts, so NTM accepts.
If $w \notin L$, all choices of $a, b$ reject, so NTM does not accept.

This NTM is a decider because it halts on every computational branch.
Simulating NTMs

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

**Proof idea:** “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 the configurations at depth 1, then all the configurations at depth 2, etc.

c) Either will always work
Simulating TMs

Theorem: Every nondeterministic TM can be simulated by an equivalent deterministic TM

Proof idea:

Breadth-first search:
Systematically try all 1-step computations, all 2-step computations, ... and see if one of them accepts
Nondeterministic TMs

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

**Proof idea:** Simulate an NTM $N$ using a 3-tape TM

(See Sipser for full description)
TM

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**

**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!

"Meta-mathematical"
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?

Question: Can computers automate mathematicians?

Question: How automatable are the tasks we saw in language theory?
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

\[ q_0 : 0, \quad q_1 : 1 \]

\[ \Sigma = \{ a, b \} \quad \delta_a = 00 \quad \delta_b = 11 \]

\[ S(q_0,a) = q_0, \ldots \]

\[ L \Omega = 0,1 \# 00,11 \# (0,00,1), (0,11,0), (1,00,0), (1,11,1) \]

\[ \# 0 \# 0 \]

\[ \delta \]
Representation independence

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

Let \[
\cdot \\]
be a different encoding scheme

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

- Decide if \([0, w] \) accepts \(D \) if \(D \) which accepts \( w \) as follows:
  1) Convert encoding \([n, w]\) to \(\langle 0, w \rangle\)
  2) Give \(\langle 0, w \rangle\), determine if \(D \) accepts \( w \)

We’ll take \( \langle \cdot \rangle \) 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 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 $A_{NFA} = \{\langle N, w \rangle \mid \text{NFA } N \text{ accepts } w \}$?

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) 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 \( A_{\text{DFA}} \) on input \( \langle D, w \rangle \)
2. Accept if the decider accepts; reject otherwise

**Analysis:**
- If \( w \in L \), \( \langle 0, w \rangle \rightarrow A_{\text{DFA}} \rightarrow \text{decider accepts} \)
- If \( w \notin L \), \( \langle 0, w \rangle \rightarrow A_{\text{DFA}} \rightarrow \text{decider rejects} \)
Classes of Languages

- Regular
- Decidable
- Recognizable