Lecture 11:

- More TM Variants and Closure Properties
- Church-Turing Thesis

Reading:
Sipser Ch 3.2

Mark Bun
October 14, 2021
TM Variants
TM 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

...
Multi-Tape TMs

Fixed number of tapes $k$  

Transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$  

$k$ can’t depend on input or change during computation
How to Simulate It

To show that a TM variant is no more powerful than the basic, single-tape TM:

Show that if $M$ is any variant machine, there exists a basic, single-tape TM $M'$ that can simulate $M$

(Usual) parts of the simulation:

• Describe how to initialize the tape(s) of $M'$ based on the input to $M$

• Describe how to simulate one step of $M$’s computation using (possibly many steps of) $M'$
Multi-Tape TMs are Equivalent to Single-Tape TMs

**Theorem:** Every $k$-tape TM $M$ with can be simulated by an equivalent single-tape TM $M'$.
Simulating Multiple Tapes

Implementation-Level Description of $M'$

On input $w = w_1 w_2 \ldots w_n$
1. Format tape into $\# w_1 w_2 \ldots w_n \# \# \# \# \# \# \# \#$
2. For each move of $M$:
   - Scan left-to-right, finding current symbols
   - Scan left-to-right, writing new symbols,
   - Scan left-to-right, moving each tape head

   If a tape head goes off the right end, insert blank
   If a tape head goes off left end, move back right
Why are Multi-Tape TMs Helpful?

To show a language is Turing-recognizable or decidable, it’s enough to construct a multi-tape TM

Often easier to construct multi-tape TMs

**Ex.** Decider for \( \{a^i b^j | i > j \} \)

On input \( w \):

1) Scan tape 1 left-to-right to check that \( w \in L(a^* b^*) \)

2) Scan tape 2 left-to-right to copy all \( b \)'s to tape 2

3) Starting from left ends of tapes 1 and 2, scan both tapes to check that every \( b \) on tape 2 has an accompanying \( a \) on tape 1. If not, reject.

4) Check that the first blank on tape 2 has an accompanying \( a \) on tape 1. If so, accept; otherwise, reject.
Why are Multi-Tape TMs Helpful?

To show a language is Turing-recognizable or decidable, it’s enough to construct a multi-tape TM

Very helpful for proving **closure properties**

**Ex.** Closure of recognizable languages under union. Suppose $M_1$ is a single-tape TM recognizing $L_1$, $M_2$ is a single-tape TM recognizing $L_2$

On input $w$:

1) Scan tapes 1, 2, and 3 left-to-right to copy $w$ to tapes 2 and 3

2) Repeat forever:

   a) Run $M_1$ for one step on tape 2

   b) Run $M_2$ for one step on tape 3

   c) If either machine accepts, **accept**

   Interlaced computation deals w/ possibility that $M_1$ loops on $w$, while $M_2$ accepts $w$. 

10/14/2021 CS332 - Theory of Computation
Closure Properties

The Turing-decidable languages are closed under:

- Union
- Concatenation
- Star
- Intersection
- Reverse
- Complement

The Turing-recognizable languages are closed under:

- Union
- Concatenation
- Star
- Intersection
- Reverse
- Complement

\[
\text{L \ decidable, } \exists \ TM \ M \ s.t. \ \forall w:\
\begin{align*}
\text{w} \in L & \Rightarrow M \text{ accepts } w \\
\text{w} \notin L & \Rightarrow M \text{ rejects } w
\end{align*}
\]

\[
\text{M flips between accept/reject states of M}
\]

\[
\text{L \ recognizable } \Rightarrow \exists \ TM \ M \ s.t. \ \forall w:\
\begin{align*}
\text{w} \in L & \Rightarrow M \text{ accepts } w \\
\text{w} \notin L & \Rightarrow M \text{ rejects } w
\end{align*}
\]

\[
\text{Not complement: Above construction fails because it may loop on strings it should accept. Label: we'll see it recognizable } L \ s.t. \ L \text{ is not recognizable}
\]
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

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

Input abba

Computation path 1:
q₀ abba
q₀ q₁ ba
q₁ b q₂ a
q₂ a q₃ a
q₃ a q₄ a
q₄ accept

Computation path 2:
q₀ abba
q₀ q₁ ba
q₁ b q₂ a
q₂ b xa
q₃ q₄ a
q₄ a q₅ a
q₅ a q₆ a
q₆ a q₇ a
q₇ a q₈ a
q₈ abba
q₈ accept
Nondeterministic TMs

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

Implementation-Level Description:

On input string \( w \) (over alphabet \( \{a,b\} \))

1) Scan tape left to right, at some point (nondeterministically) goto step 2

2) a) Read next symbol \( S \), cross it off

   b) Move head left repeatedly, looking for \( S \). If found, cross it off.
      (otherwise reject)

   c) Move read right until reaches non-\( X \) symbol. If \( U \) hit, go to step 3

   d) Repeat (go back to 2 a)

3) Check that entire tape is \( X \)'s. If so accept.
Nondeterministic TMs

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

What is the language recognized by this NTM?

a) \( \{ \text{ww} \mid w \in \{a, b\}^* \} \)

b) \( \{ \text{ww}^R \mid w \in \{a, b\}^* \} \)

c) \( \{ \text{ww} \mid w \in \{a, b, x\}^* \} \)

d) \( \{ \text{wx}^n w^R \mid w \in \{a, b\}^*, n \geq 0 \} \)
Nondeterministic TMs

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

**Implementation level:**

On input $w$:

1) Nondeterministically try:
   a) Run $M_1$ on tape, accept if accepts, or
   b) Run $M_2$ on tape, accept if accepts.
Nondeterministic TMs

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

High-Level Description:

On input \( w \):

1) Nondeterministically "guess" \( a \in \{2, \ldots, w^3\} \) and \( b \in \{2, \ldots, w^3\} \)

2) Multiply \( a \cdot b \), check equal to \( w \). Accept if so, reject o.w.

Analysis:

- If \( w \in L \), \( \exists a, b \in \{2, \ldots, w^3\} \) s.t. \( a \cdot b = w \). So path of computation that guessed these factors leads to accept.
- If \( w \notin L \), every \( a, b \) guessed will lead to reject.
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 $N$ recognizes language $L$ if:

- $w \in L \Rightarrow \exists$ computation path of $N$ on $w$ leading to accept
- $w \notin L \Rightarrow$ every computation path leads to reject, looping or failure

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

$N$ decides $L$ if:

- $w \in L \Rightarrow \exists$ computation path leading to accept
- $w \notin L \Rightarrow$ every computation path leads to reject (must halt)
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)
TMIs are equivalent to...

- TMIs with “stay put”
- TMIs with 2-way infinite tapes
- Multi-tape TMIs
- Nondeterministic TMIs
- Random access TMIs
- 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

*Normative, prescriptive*

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

*Empirical*

The Church-Turing Thesis is not a mathematical statement! Can’t be mathematically proved

“Not too mathematical”