Lecture 11:

- TM Variants
- Closure Properties

Reading:
Sipser Ch 3.2

Mark Bun
March 1, 2020
The Basic Turing Machine (TM)

- Input is written on an infinitely long tape
- Head can both read and write, and move in both directions
- Computation halts when control reaches “accept” or “reject” state
Example

$q_0 \rightarrow 0 \rightarrow 0, R$
$q_1 \rightarrow \perp \rightarrow \perp, R$
$q_{\text{reject}} \rightarrow \perp \rightarrow \perp, R$
$q_0 \rightarrow \perp \rightarrow \perp, R$
$q_1 \rightarrow 0 \rightarrow 0, R$
$q_{\text{accept}}$
Formal Definition of a TM

A TM is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet (does not include $\sqcup$)
- $\Gamma$ is the tape alphabet (contains $\sqcup$ and $\Sigma$)
- $\delta$ is the transition function

...more on this later

- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state ($q_{\text{reject}} \neq q_{\text{accept}}$)
TM Transition Function

\[ \delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \]

\( L \) means “move left” and \( R \) means “move right”

\[ \delta(p, a) = (q, b, R) \] means:

- Replace \( a \) with \( b \) in current cell
- Transition from state \( p \) to state \( q \)
- Move tape head right

\[ \delta(p, a) = (q, b, L) \] means:

- Replace \( a \) with \( b \) in current cell
- Transition from state \( p \) to state \( q \)
- Move tape head left UNLESS we are at left end of tape, in which case don’t move
Configuration of a TM

A string with captures the state of a TM together with the contents of the tape

\[ q_5 \]

\[ \begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 1 & 1 \\
\end{array} \]
Configuration of a TM: Formally

A configuration is a string $uqv$ where $q \in Q$ and $u, v \in \Gamma^*$

- Tape contents = $uv$ (followed by blanks $\sqcup$)
- Current state = $q$
- Tape head on first symbol of $v$
How a TM Computes

Start configuration: $q_0w$

One step of computation:
- $ua q bv$ yields $uac q' v$ if $\delta(q, b) = (q', c, R)$
- $ua q bv$ yields $u q' acv$ if $\delta(q, b) = (q', c, L)$
- $q bv$ yields $q' cv$ if $\delta(q, b) = (q', c, L)$

Accepting configuration: $q = q_{\text{accept}}$

Rejecting configuration: $q = q_{\text{reject}}$
How a TM Computes

$M$ accepts input $w$ if there is a sequence of configurations $C_1, \ldots , C_k$ such that:

- $C_1 = q_0w$
- $C_i$ yields $C_{i+1}$ for every $i$
- $C_k$ is an accepting configuration

$L(M) = $ the set of all strings $w$ which $M$ accepts

$A$ is Turing-recognizable if $A = L(M)$ for some TM $M$:

- $w \in A \Rightarrow M$ halts on $w$ in state $q_{\text{accept}}$
- $w \notin A \Rightarrow M$ halts on $w$ in state $q_{\text{reject}}$ OR $M$ runs forever
Recognizers vs. Deciders

\(L(M) = \) the set of all strings \(w\) which \(M\) accepts

\(A\) is Turing-recognizable if \(A = L(M)\) for some TM \(M\):

- \(w \in A \Rightarrow M\) halts on \(w\) in state \(q_{\text{accept}}\)
- \(w \notin A \Rightarrow M\) halts on \(w\) in state \(q_{\text{reject}}\) OR \(M\) runs forever

\(A\) is (Turing-)decidable if \(A = L(M)\) for some TM \(M\) which halts on every input

- \(w \in A \Rightarrow M\) halts on \(w\) in state \(q_{\text{accept}}\)
- \(w \notin A \Rightarrow M\) halts on \(w\) in state \(q_{\text{reject}}\)
Back to Hilbert’s Tenth Problem

Computational Problem: Given a Diophantine equation, does it have a solution over the integers?

$L =$

- $L$ is Turing-recognizable
- $L$ is not decidable (1949-70)
TM Variants
How Robust is the TM Model?

Does changing the model result in different languages being recognizable / decidable?

So far we’ve seen...
- We can require that FAs/PDAs have a single accept state
- (CFGs can always be put in Chomsky Normal Form)
- Adding nondeterminism does not change the languages recognized by finite automata

Turing machines have an **astonishing** level of robustness
Extensions that do not increase the power of the TM model

- TMs that are allowed to “stay put” instead of moving left or right

\[ \delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\} \]

**Proof** that TMs with “stay put” are no more powerful:

**Simulation:** Convert any TM \( M \) with “stay put” into an equivalent TM \( M' \) without

Replace every “stay put” instruction in \( M \) with a move right instruction, followed by a move left instruction in \( M' \)
Extensions that do not increase the power of the TM model

- TMs with a 2-way infinite tape, unbounded left to right

**Proof** that TMs with 2-way infinite tapes are no more powerful:

**Simulation:** Convert any TM $M$ with 2-way infinite tape into a 1-way infinite TM $M'$ with a “two-track tape”
Formalizing the Simulation

\[ M' = (Q', \Sigma, \Gamma', \delta', q'_0, q'_{\text{accept}}, q'_{\text{reject}}) \]

New tape alphabet: \[ \Gamma' = (\Gamma \times \Gamma) \cup \{\$\} \]

New state set: \[ Q' = Q \times \{+, -\} \]

\((q, -)\) means “\(q\), working on upper track”

\((q, +)\) means “\(q\), working on lower track”

New transitions:

If \( \delta(p, a_-) = (q, b, L) \), let \( \delta'((p, -), (a_-, a_+)) = ((q, -), (b, a_+), R) \)

Also need new transitions for moving right, lower track, hitting $, initializing input into 2-track format
TM\text{es 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 = 2-stack PDAs
- Primitive recursive functions
- Cellular automata

...
Church-Turing Thesis

The equivalence of these models is a mathematical theorem.

Church-Turing Thesis: Each of these models captures our intuitive notion of algorithms.

The Church-Turing Thesis is not a mathematical statement!
Multi-Tape TMs

Fixed number of tapes $k$ (can’t change during computation)
Transition function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
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

On input \( w = w_1 w_2 \ldots w_n \)
1. Format tape into \( \# \hat{w}_1 \hat{w}_2 \ldots \hat{w}_n \\# \hat{\sqcap} \# \hat{\sqcap} \# \ldots \# \)
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, suffices 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 $L_2$
Non-deterministic TMs

At any point in computation, may non-deterministically branch. Accepts iff there exists an accepting branch.

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

Ex. NTM for $\{w \mid w \text{ is a binary number representing the product of two positive integers } a, b\}$
Non-deterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

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

Finite control

Input $w$ to $N$ (read-only)

Simulation tape (run $N$ on $w$ using nondeterministic choices from tape 3)

Address in computation tree