Outline of Turing Machines and Complexity

1. Turing machine (TM) = formal model of a computer running a particular program.
   ✦ We must argue that the TM can do exactly what a computer can do, albeit slower.

2. We use the simplicity of the TM model to prove formally that there are specific problems (=languages) that the TM cannot solve.
   ✦ Two classes: “recursively enumerable”
     = TM can accept the strings in the language but cannot tell for certain that a string is not in the language; “non-RE” = no TM can even recognize the members of the language in the RE sense.

3. We then look at problems (languages) that do have TM’s that accept them and always halt; i.e., they not only recognize the strings in the language, but they tell us when they are sure the string is not in the language.
   ✦ The classes P and NP are those languages recognizable by deterministic (resp., nondeterministic) TM’s that halt within a time that is some polynomial in the input.
   ✦ Polynomial is as close as we can get, because real computers and different models of (deterministic) TM’s can differ in their running time by a polynomial function, e.g., a problem might take \( O(n^2) \) time on a real computer and \( O(n^6) \) on a TM.

4. NP-complete problems: Since we don’t know whether \( P = \mathcal{NP} \), but it appears that at least some problems in \( \mathcal{NP} \) take exponential time, the best we can do is show that a certain problem is “NP-complete,” = if this problem is in \( P \), then all of \( \mathcal{NP} \) is in \( P \).

5. Some specific problems that are NP-complete: satisfiability of boolean (propositional logic) formulas, traveling salesman, etc.

Intuitive Argument About an Undecidable Problem

Given a C program, does it print `hello, world` as the first 13 characters of output?
• We prove there is no C program to solve that problem by supposing that there were such a program \( H \), the “hello-world-tester.”
  
  - \( H \) takes as input a C program \( P \) and an input file \( I \) for that program, and tells whether \( P \), with input \( I \), “prints hello world” (by which we mean it does so as the first 13 characters).

• Modify \( H \) to a new program \( H_1 \) that acts like \( H \), but when \( H \) prints no, \( H_1 \) prints hello, world.
  
  - Requires some thought: we need to find where no is printed and change the printf statement.

• Modify \( H_1 \) to \( H_2 \). This program takes only one input, \( P \), and acts like \( H_1 \) with both its program and data inputs equal to \( P \).
  
  - I.e., \( H_2(P) = H_1(P, P) \).
  
  - Requires more thought: \( H_2 \) must buffer its input so it can be used as both the \( P \) and \( I \) inputs to \( H_1 \).

• \( H_2 \) cannot exist. If it did, what would \( H_2(H_2) \) do?
  
  - If \( H_2(H_2) = \text{yes} \), then \( H_2 \) given \( H_2 \) as input evidently does not print hello, world. But \( H_2(H_2) = H_1(H_2, H_2) = H(H_2, H_2) \), and \( H_1 \) prints yes if and only if its first input, given its second input as data, prints hello, world. Thus, \( H_2(H_2) = \text{yes} \) implies \( H_2(H_2) = \text{hello, world} \).
  
  - But if \( H_2(H_2) = \text{hello, world} \), then \( H_1(H_2, H_2) = \text{hello, world} \) and \( H(H_2, H_2) = \text{no} \). Thus, \( H_2(H_2) = \text{hello, world} \) implies \( H_2(H_2) \neq \text{hello, world} \).

The TM

• Finite-state control, like PDA.

• One read-write tape serves as both input and unbounded storage device.

  - Tape divided into cells.

  - Each tape holds one symbol from the tape alphabet.

  - Tape is “semi-infinite”; it ends only at the left.
- **Tape head** marks the “current” cell, which is the only cell that can influence the move of the TM.

- Initially, tape holds $a_1 a_2 \cdots a_n B B \cdots$ where $a_1 a_2 \cdots a_n$ is the *input*, chosen from an *input alphabet* (subset of the tape alphabet) and $B$ is the *blank*.

**Formal TM**

$$M = (Q, \Sigma, \delta, q_0, B, F),$$

where:

- $Q =$ finite set of states.
- $\Sigma =$ tape alphabet; $\Sigma \subseteq \Sigma =$ input alphabet.
- $B$ in $\Sigma = \text{blank}$.
- $q_0$ in $Q =$ start symbol; $F \subseteq Q =$ accepting states.
- $\delta$ takes a state and tape symbol, returns a new state, replacement symbol (either might not change) and a direction $L/R$ for head motion.

**Example**

Nontrivial examples are hard to come by. Here’s a TM that checks its third symbol is 0, accepts if so, and runs forever, if not.

$$M = (\{ p, q, r, s, t \}, \{0, 1\}, \{0, 1, B\}, p, B, \{s\})$$

1. $\delta(p, X) = (q, X, R)$ for $X = 0, 1$.
2. $\delta(q, X) = (r, X, R)$ for $X = 0, 1$.
3. $\delta(r, 0) = (s, 0, L)$.
4. $\delta(r, 1) = (t, 1, R)$.
5. $\delta(t, X) = (t, X, R)$ for $X = 0, 1, B$.

**ID’s of a Turing Machine**

The ID (instantaneous description) captures what is going on at any moment: the current state, the contents of the tape, and the position of the tape head.

- Keep things finite by dropping all symbols to the right of the head and to the right of the rightmost nonblank.

  - Subtle point: although there is no limit on how far right the head may move and write nonblanks, at any finite time, the TM has visited only a finite prefix of the infinite tape.
• Notation: $a q \beta$ says:
  ✦ $a$ is the tape contents to the left of the head.
  ✦ The state is $q$.
  ✦ $\beta$ is the nonblank tape contents at or to the right of the tape head.
• One move indicated by $\vdash$; zero, one, or more moves represented by $\vdash^*$.
• Check the reader for the detailed definition of $\vdash$.

Example
With input 0101, the sequence of ID’s of the TM is: $p0101 \vdash 0q101 \vdash 01r01 \vdash 0s101$.
• At that point it halts, since state $s$ has no move when the head is scanning 1.
With input 0111 the sequence is: $p0111 \vdash 0q111 \vdash 01r11 \vdash 011r1 \vdash 0111rt \vdash \cdots$.
• The TM never halts, but continues to move right.

Acceptance by Final State and by Halting
One way to define the language of a TM is by the set of input strings that cause it to reach an accepting state.
• $L(M) = \{ w \mid q_0 w \vdash^* \alpha p \beta \text{ for some } p \in F \text{ and any } \alpha \text{ and } \beta \text{ in } ,^* \}$.
Another way is to define the set of strings that cause the TM to halt = have no next move.
• $H(M) = \{ w \mid q_0 w \vdash^* \alpha p X \beta, \text{ and } \delta(p, X) \text{ is not defined} \}$.
• Subtle point: a TM can appear to halt if the next move would take the head off the left end of the tape.
• Given any TM, we can mark the left end so that never happens; i.e., we produce a modified TM that accepts the same language and halts rather than fall off the left end.

Example
• The TM $M$ of our previous example has $L(M)$ equal to those strings in the language of RE $(0 + 1)(0 + 1)(0 + 1)^*$.
• $H(M)$ is the language of $\epsilon + 0 + 1 + (0 + 1)(0 + 1) + (0 + 1)(0 + 1)(0 + 1)^*$. 

**Equivalence of Acceptance by Final State and Halting**

We need to show $L$ is $L(M_1)$ for some TM $M_1$ if and only if $L$ is $H(M_2)$ for some TM $M_2$.

If

Modify $M_2$ as follows:

1. Introduce one accepting state $r$.

2. Whenever there is no transition for $M_2$ on state $q$ and symbol $X$, add a transition to state $r$, moving right (so we can’t possibly fall off the left end) and leaving symbol $X$.

Only-If

Roughly, we let $M_2$ simulate $M_1$, but if $M_1$ enters an accepting state, $M_2$ has no next move and so halts.

• Major problem: $M_1$ could halt without accepting.
  ✦ To avoid this problem, introduce state $r$ that moves right on every symbol, staying in state $r$ and leaving the tape symbols unchanged.
  ✦ Give $M_2$ a transition to $r$ (moving right) on every state-symbol combination that does not have a rule.

• Also, remove all transitions where the state is an accepting state of $M_1$, so $M_2$ will halt in those situations.

**Falling Off the Left End of Tape**

The reader talks about the funny situation where the TM would halt but falls off the left end of tape.

• This situation is not halting.

• Neither does a TM accept if it tries to enter an accepting state as it falls off the left end.

• We can prevent falling off the left end, by marking the leftmost cell, as in the reader.

• But it appears we do not need to do so in order to prove the equivalence of halting/accepting, since neither occurs when the TM falls off the left end.