1 Turing Machines (3.1, Continued)

We continue our notes on Turing machines.

1.1 Describing Turing Machines

There are several ways we can describe Turing machines.

- **Formal Definition:** A set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state. Essentially, drawing out the Turing machine. This is the most detailed level of description.
- **Implementation-Level Definition:** English prose to describe Turing machine head movements relative to the contents of the tape; in other words, we use English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape. Note that we do not give details of states or transition function.
- **High-Level Description:** A description of the algorithm, without the implementation details of a machine. As part of this description, we can "call" and run another Turing machine as a subroutine. Here, we do not need to mention how the machine manages its tape or head.

1.1.1 Example: Describing Turing Machines

Give an *implementation-level* description of a Turing machine which *decides* the empty set.

Let M be the Turing machine where, on some input w, we reject.

1.1.2 Example: Describing Turing Machines

Give a *high-level* description of a Turing machine which *decides* the language

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

Here, we say that $\langle G \rangle$ is the encoding of the graph. For example, if G is defined by

$$G = (V, E)$$

where

$$V = \{1, 2, 3, 4\} \qquad E = \{(1, 2), (2, 3), (3, 1), (1, 4)\}$$

then the encoding is given by

$$\langle G \rangle = \texttt{(1,2,3,4)((1,2),(2,3),(3,1),(1,4))}$$

Let M be the Turing machine where, on input $\langle G \rangle$ (the encoding of the graph), we do the following:

1. Select the first node of G and mark it.
2. While we still have nodes to mark, for each node in G, mark it if it is attached by an edge to a node that is already marked.
3. Scan all the nodes of G to determine whether they are all marked. If they are, *accept*. Otherwise, *reject*.

1.2 Class of Recognizable and Decidable Languages

The class of decidable *and* recognizable languages is closed under:

- Union.
- Concatenation.
- Intersection.
- Kleene Star.

The class of *decidable* languages is also closed under complementation.

1.2.1 Class of Decidable Languages Closed Under Union

Theorem 1.1

The class of decidable languages over a fixed alphabet Σ is closed under union.

Proof. Let L_1 and l_2 be languages over Σ and suppose M_1 and M_2 are Turing machines that decide these languages, respectively. We will define a new Turing machine, M, via a high-level description. For some input w, the Turing machine M will do the following:

1. Run M_1 on input w. If M_1 accepts w, then accept. Otherwise, go to the next step.
2. Run M_2 on input w. If M_2 accepts w, then accept. Otherwise, we reject.

To show correctness, we need to show that $L(M) = L_1 \cup L_2$ and that M is a decider. For some string s, if L_1 or L_2 contains s, then clearly the machine must recognize it. We know this is the case because our machine runs M_1 and then runs M_2 if M_1 doesn't yield an *accept* state. Now, if M_1 and M_2 both reject, then clearly s must not be in any of the two languages, and will thus not be in M. To show that M is a decider, we note that M will either accept (if s is accepted by any of the two machines) or reject if both M_1 and M_2 reject. So, there's no chance that it will loop. □

2 Variants of Turing Machines (3.2)

There are, of course, different variations of Turing machines. Some of these variants may have multiple tapes, or may use nondeterminism. The original Turing machine and its variants all have the same power in the sense taht they recognize the same class of languages.

2.1 Multitape Turing Machine

A **multitape Turing machine** is like an ordinary Turing machine with several tapes. Each tape has its own head for reading and writing. Initially, the input appears on the first tape, and the other tapes start out blank. In order to accomodate for the fact that there are multiple tapes, we now introduce a new transition function which allows for reading, writing, and moving the heads on some or all of the tapes simultaneously:

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R, S\}^k$$

where k is the number of tapes. So, it follows that the expression

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

means that if the machine is in state q_i and heads 1 through k are reading the symbols a_1 through a_k, the machine goes to state q_j and writes symbols b_1 through b_k, and directs each head to move left or right (or stay put, S), as specified.

2.1.1 Multitape vs. Single-tape Turing Machines

Theorem 2.1

Every multitape Turing machine has an equivalent single-tape Turing machine.

Remark: This is Theorem 3.13 in the textbook. The proof can be found there.

(Idea.) To show equivalence, we need to do the following:

- Given a Turing machine, we can build a multitape Turing machine which recognizes the same language.
- Given a k-tape Turing machine, we can build a one-tape Turing machine recognizing the same language.

To do this, we can make use of a delimiter to keep tape contents separate, and then use a special symbol to indicate the location of each read/write head.

2.1.2 Turing-Recognizability

Corollary 2.1

A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

Proof. A Turing-recognizable language is recognized by an ordinary (single-tape) Turing machine, which is a special case of a multitape Turing machine. This proves one direction of the corollary. The other direction comes as part of the proof of the above theorem. □

2.2 Nondeterministic Turing Machines

A **nondeterministic Turing machine** is defined in the expected way. At any point in a computation, the machine can proceed according to several possibilities. The transition function, then, has the form

$$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Like a nondeterministic finite automaton, if a computation ends up at an accept state, then the machine as a whole accepts the input. If, after going through all possibilities, the machine doesn't reach an accept state, then the machine rejects the input.

2.2.1 Nondeterministic vs. Deterministic Turing Machines

Theorem 2.2

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof. (Idea.) For the proof of equivalence, we need to show two things.

1. Given a Turing machine, we need to build a nondeterminism Turing machine recognizing the same language.
2. Given a nondeterminism Turing machine, we need to build a deterministic Turing machine recognizing the same language.

To do this, we can try all possible branches of the nondeterministic computation. We have three tapes: a "read-only" input tape, a simulation tape, and a tape tracking nondeterminism branching. □

2.2.2 Turing-Recognizability

Corollary 2.2

A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

Proof. Any deterministic Turing machine is automatically a nondeterministic Turing machine, so one direction of the corollary follows immediately. For the other direction, we can consider the proof of the theorem above. □

2.2.3 Turing-Decidable

The proof of the theorem can be modifier so that if the nondeterministic Turing machine always halts on all branches of its computation, then its corresponding deterministic Turing mahcine will as well.

Corollary 2.3

A language is decidable if and only if some nondeterministic Turing machine recognizes it.

2.3 Enumerators

Loosely defined, an enumerator is a Turing machine with an attached printer. Then, the Turing machine can use that printer as an output device to print strings out. Every time the Turing machine wants to ad a string to the list, it sends that string to the printer.

An enumerator E starts with a blank tape on its work tape. Now, if the enumerator doesn't half, then it may print an infinite list of strings. The language enumerated by E is the collection of all the strings that it

will eventually print out. Moreover, E may generate the strings of the langauge in *any order*, possibly *with repetition*.

For any given Σ, there is an enumerator whose language is the set of all strings over Σ. To see this, we do standard string ordering. In particular:

- Order strings first by length.
- Then, by dictionary order.

2.3.1 Turing-Recognizability

Theorem 2.3

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof. First, we want to show that if we have an enumerator E that enumerates a language A, then a Turing machine A recognizes A. To describe M, we note that on some input w, we:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept it.

Clearly, M accepts those strings that appear on E's list. To show the other direction, note that if a Turing machine M recognizes a language A, then we can construct the following enumerator E for A. Say that $s_1, s_2, s_3 \ldots$ is a list of all possible strings in Σ^*. Then, for some input w, we can describe E like so:

1. Ignore the input.
2. Repeat the following for $i = 1, 2, 3, \ldots$:
   - Run M for i steps on each input $s_1, s_2, \ldots, s_i$.
   - If any computations accept, print out the corresponding s_j.

If M accepts a particular string w, then it will eventually appear on the list generated by E. In fact, it will appear on the list infinitely many times because M runs from the beginning on each string for each repetition of step 1. This procedures gives the effect of running M in parallel on all possible input strings. □