Recitation 04: TMs and Quiz Review

In today's recitation, we will review Turing machines and get some more practice working with them. We will then do a short review for the quiz and discuss closure properties for decidable and recognizable languages.

Turing Machine variants

We saw in class that there are several equivalent variants of Turing machines. We can use any of these equivalent formulations when proving that a language is Turing-recognizable.

Theorem 1. *The following machines are all equivalent in computational power to single-tape, deterministic TMs.*

1. *Nondeterministic TMs (NTM)*
2. *Multi-tape TMs*
3. *Enumerators (i.e. a two-tape TM where the second tape is a printer)*

To review the proofs that all of these are equivalent, see **Section 3.2** of the textbook.
Remember that an NTM accepts if *at least one* branch of its computation accepts.

Another way to formulate the above theorem is that the following four statements are equivalent (so there is an "if and only if" relationship between each pair of these statements):

1. Language B is Turing-recognizable.
2. $B = L(N)$ for some NTM N.
3. $B = L(M)$ for some multi-tape TM M.
4. $B = L(E)$ for some enumerator E.

There is a similar theorem for variants of Turing deciders.

Theorem 2. *The following machines are all equivalent in computational power to single-tape, deterministic Turing deciders.*

1. *Nondeterministic Turing deciders.*
2. *Multi-tape Turing deciders.*

A nondeterministic Turing decider must halt on *every branch* of its computation. It accepts if *at least one* branch of its computation accepts.

3. *Enumerators that enumerate strings in lexicographic order.*

#3 is proved as problem 4 in pset 2(a language is decidable if and only if there is an enumerator that enumerates its language in lexicographic order).

Example 1. *Prove that there's an enumerator E that generates a language if and only if there's a Turing machine M that recognizes. (A subset of theorem 1)*

Solution. The standard way to solve this kind of problem is to use each of the machines to construct the other. To construct M using E, we have M run E. If at any point E outputs the input, M accepts. (M rejects all strings not in the language by looping)

The main subtlety is in using M to construct an enumerator, this requires two standard tricks:

1. Simulating multiple instances of M running on different input in a single machine, by repeatedly running a step in each. (to avoid getting stuck in one that never halts). This is similar to the proof that none-deterministic machines can not recognize that deterministic Turing machine can not recognize. But as a reminder it goes as follows:

   (a) Separate your tape into finite regions separated by #s so that each machine gets a region. If a machine tries to read outside is region while being simulated, we exit simulation and shift everything to the right to grow it's the region.

   (b) Store the current state of the simulated machine at the beginning of each region.

   (c) Denoted where the simulated head is by putting a dot on it's position. Each region should have one and only one dot.

2. Doing the above, but adding a new thread simulating M on a new word after each iteration on all the existing threads. Note that every machine gets an infinite number of steps. Once any of the simulated machines accepts, its string is printed.

Next, let's work through an example showing that TMs are also computationally equivalent to a "broken" TM where trying to move left would move the tape head all the way to the left. [1]

[1] Some of the recitations this week might not have had time to do this example.

Example 2. *Let a **left-reset TM** be a TM, but when the transition function returns an L, the TM actually goes all the way to the left instead of moving left by one cell. Then left-reset TMs are equivalent to standard TMs.*

Solution. We can prove that left-reset TMs are equivalent to standard TMs by showing that left-reset TMs can simulate standard TMs. Since the only difference in left-reset TMs is the moving left operation, it is enough to show that left-reset TMs can simulate moving left by one cell. There are two ways of showing this:

1. To move left by one cell, the left-reset TM shifts every symbol right by one. More specifically, the left-reset TM does the following to move left one cell:

   (a) Mark the current cell.

   (b) Move all the way to the left.

   (c) Shift every symbol on the tape right by one cell, taking care to not actually shift the mark.

   (d) Move all the way to the left.

   (e) Go right until we reach the mark.

   Another detail to be careful about is how the left-reset TM knows where the tape symbols end. This can be resolved by keeping a mark at the rightmost cell ever visited.

2. To move left by one cell, the left-reset TM makes a mark on the current cell, and advances using another mark slowly to get to the cell before the first marked cell. More specifically, the left-reset TM does the following to move left one cell:

   (a) Mark the current cell with a $\bullet$.

   (b) Go all the way to the left and mark the leftmost cell with a $\star$.

   (c) Repeat the following loop:

      i. Go to the left and scan right until finding the $\star$.

      ii. Move right one cell.

      iii. If the cell has a $\bullet$, remove it and go back to the $\star$ and change it to a $\bullet$ exit the loop. Otherwise, the $\star$ is not left of the original cell. Advance the $\star$ by one cell.

   (d) Go to the $\star$ and remove it. The left-reset TM should be on the cell one left of the original cell.

T-recognizable and decidable languages

Turing machines are stronger than the other models of computation we have seen in class (such as finite automata or PDAs). In fact, the **Church-Turing thesis** says that any real-world algorithm can be computed by a Turing machine.

For a quick review of Turing machines and the definitions of Turing recognizable and Turing decidable, read **Section 3.1** of the textbook.

To show that a language B is **Turing-recognizable**, we will want to construct a TM M that recognizes B. This means that M accepts every string in B, and either enters the reject state or loops forever on strings not in B.

To show that a language B is **Turing-decidable**, we will want to construct a TM M that *decides* B. This means that M accepts every

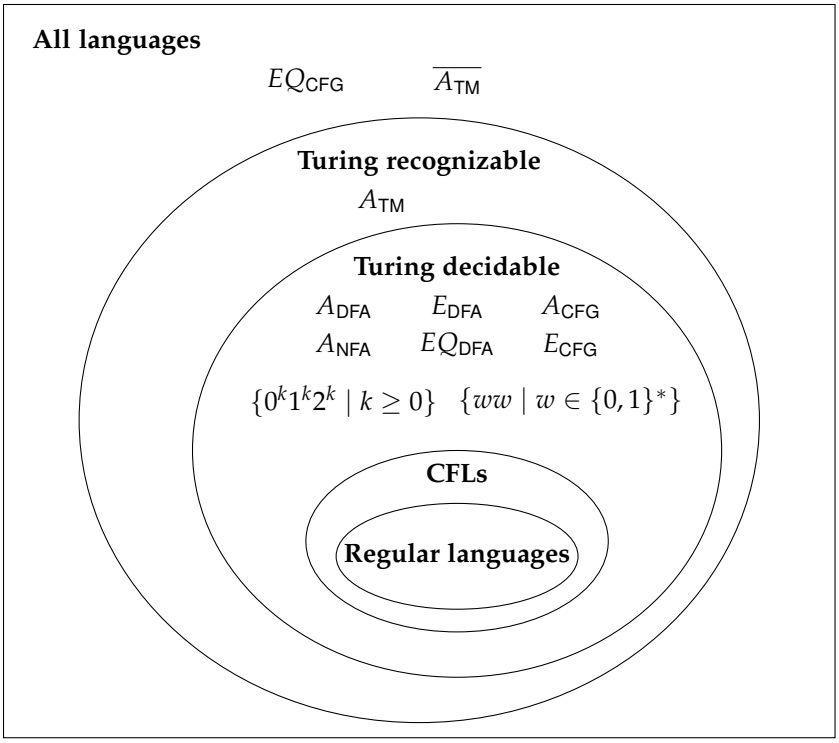

Figure 1: Examples of Turing decidable and Turing recognizable languages from class.

string in B, and enters the reject state and halt on strings not in B. We will need to make sure that M always halts, and does not loop forever on any input.

Note that every Turing-decidable language is Turing-recognizable, since every Turing decider is a TM.

We have seen several examples of Turing decidable languages in class, listed in Figure 1. We can use the deciders for these languages as subroutines to prove that other languages are decidable. We will see this in the examples. We have also seen one example of a language that is Turing recognizable but not decidable:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$$

There are also some languages such as EQ_{CFG} and $\overline{A_{TM}}$ that are not T-recognizable. We will learn more about undecidable and unrecognizable languages next week.

Quiz Review

So far, we've covered three main models of computation – finite automata, PDAs, and Turing machines. The languages recognized by these models are summarized in the following table.

X	Recognizes X	Generates X
Regular	DFA/NFA	Regular expression
CFL	(non-det.) PDA	CFG
T-decidable	Turing decider	Enumerator in string order
T-recognizable	Turing machine	Enumerator

Table 1: Summary of what recognizes/generates the language classes: regular, context-free, decidable, and Turing-recognizable.

Closure properties for T-recognizable and T-decidable languages

Turing-decidable languages are closed under union, concatenation, star, intersection, and complement.

Turing-recognizable languages are closed under union, concatenation, star, and intersection. Here is a counterexample showing that T-recognizable languages are *not* closed under complement: A_{TM} is T-recognizable, but we will see later in class that $\overline{A_{TM}}$ is T-unrecognizable.

Theorem 3. *Turing-decidable languages are closed under union, concatenation, star, intersection, and complement.*

Proof. Suppose A and B are Turing-decidable languages. Then there exists a TM M_A that decides A and a TM M_B that decides B. Since M_A and M_B are deciders, they are guaranteed to halt on any input (either accepting or rejecting by halting).

Union. We construct a Turing decider $M_\cup$ for $A \cup B$.

On input string w:

1. Run M_A on w.
2. Run M_B on w.
3. **Accept** if either M_A or M_B accepts. **Reject** if both reject.

Concatenation. We construct a Turing decider $M_\circ$ for $A \circ B$.

On input w:

1. Nondeterministically guess where we split w into two strings: $w = w_1 w_2$.
2. Run M_A on w_1.
3. Run M_B on w_2.
4. **Accept** if on some branch of the computation, both M_A and M_B accept. **Reject** otherwise.

Star. We construct a Turing decider M_* for A^*.

On input w:

1. Nondeterministically guess the number k of partitions. Then guess where we split w into k strings: $w = w_1 w_2 \dots w_k$.
2. Sequentially run M_A on $w_1, w_2, \dots, w_k$.
3. **Accept** if on some branch of the computation, M_A accepts on all the strings. **Reject** otherwise.

Intersection. We construct a Turing decider $M_\cap$ for $A \cap B$.

On input string w:

1. Run M_A on w.
2. Run M_B on w.
3. **Accept** if both M_A or M_B accept. **Reject** if either reject.

Complement. We construct a Turing decider M' for $\overline{A}$.

On input string w:

1. Run M_A on w.
2. **Accept** if M_A rejects. **Reject** if M_A accepts.

The above algorithm for M' would not work if A were Turing-recognizable but not decidable, because M_A might reject by looping forever on w. Then M' would not halt and accept w, even though w is in $\overline{A}$.

Theorem 4. *Turing-recognizable languages are closed under union, concatenation, star, and intersection.*

Proof. Suppose A and B are Turing-recognizable languages. Then there exists a TM M_A that recognizes A and a TM M_B that recognizes B. Since M_A and M_B are TMs, they can accept, reject by halting, or reject by looping.

Union. We construct a TM $M_\cup$ that recognizes $A \cup B$. We need to slightly modify our proof for Turing-decidable languages. Since M_A or M_B might reject by looping forever, we have to run the two machines in parallel on the input (rather than sequentially).

On input string w:

1. Run M_A and M_B on w in parallel.
2. **Accept** if either M_A or M_B accepts. **Reject** otherwise.

Concatenation. Same algorithm as the proof for Turing-decidable languages. We can run the machines sequentially as before. Note that

if M_A or M_B (or both) reject by looping, then the new machine also rejects by looping (in that branch). This is okay because in any given branch, we want to reject if either machine rejects on its section of w.

Star. Same algorithm as the proof for Turing-decidable languages. We can run M_A sequentially on the strings $w_1, w_2, \ldots w_k$ as before. Note that if M_A rejects by looping on one of the strings w_i, then the new machine rejects the whole string $w = w_1 w_2 \ldots w_k$ by looping (in that branch). This is okay because in any given branch, we want to reject if M_A rejects any of w_1, w_2, …, w_k.

Intersection. Same algorithm as the proof for Turing-decidable languages. Note that if M_A or M_B (or both) reject by looping, then our machine $M_\cap$ would also reject by looping. This is okay because we want to reject if M_A or M_B rejects anyway.

The following table summarizes important closure properties for the classes of languages we have learned about.

class	A^*	$A \circ B$	$A \cup B$	$A \cap B$	$\overline{A}$	$A \setminus B$
regular	yes	yes	yes	yes	yes	yes
context-free	yes	yes	yes	no†	no	no
decidable	yes	yes	yes	yes	yes	yes
recognizable	yes	yes	yes	yes	no	no

†: yes if intersected with a regular language