Recitation 04: TMs, T-recognizability, Decidability

In today’s recitation, we will gain more practice working with Turing Machines. First, we’ll review the TM variants that we saw in class this week. Then, we’ll work through a few examples for showing that a language is Turing-decidable. We will also learn about closure properties for Turing recognizable and decidable 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)

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 T-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.

Remember that an NTM accepts if at least one branch of its computation accepts.

To review the proofs that all of these are equivalent, see Section 3.2 of the textbook.

A nondeterministic Turing decider must halt on every branch of its computation. It accepts if at least one branch of its computation accepts.
2. Multi-tape Turing deciders.

3. Enumerators that enumerate strings in lexicographic order.

Let’s work through an example showing that TMs are also computationally equivalent to PDAs with two stacks. This implies that PDAs with a single stack are less powerful than PDAs with two stacks, but PDAs with two stacks are just as powerful as PDAs with three or more stacks. Adding additional stacks to the PDA beyond the second one doesn’t improve computational power, because the two-stack PDA is already equivalent to TMs which are the most powerful computational model we will see in this class. By the Church-Turing Thesis, this means PDAs with two stacks can compute any real world algorithm.

**Example 1.** PDAs with two stacks are equivalent in computational power to TMs.

**Solution 1.** First, we show that **TMs are at least as powerful as two-stack PDAs.** Let P be a two-stack PDA. We can simulate P using a Turing machine M with three tapes. The first tape of M works as the input tape of P, and the other two tapes of M serve as the two stacks. Since multitape TMs are computationally equivalent to single tape TMs, we are finished.

Next, we show that **two-stack PDAs are at least as powerful as TMs.** Let M be a Turing machine. We want to build a two-stack PDA that simulates M. Let P be a two-stack PDA with a left stack L and a right stack R. At any point in time, L contains all the tape symbols to the left of M’s head. R contains all the tape symbols to the right of M’s head. The top symbol on L is the current tape symbol that M’s head is pointing to.

We show that P can simulate moving M’s head (in either direction) and writing to the tape:

1. To simulate moving M’s head to the left, pop the top symbol of L and push it onto R.
2. To simulate moving M’s head to the right, pop the top symbol of R and push it onto L.
3. To simulate M writing on the tape to change the current symbol, pop L (equivalent to erasing M’s current symbol) and push a new symbol to L.

**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.

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 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:

\[
A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \}.
\]

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.

---

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

---

Here are some example problems for proving a language is Turing-decidable.
Example 2. Show that \( \text{ALL}_{\text{DFA}} = \{ \langle M \rangle \mid M \text{ is a DFA and } L(M) = \Sigma^* \} \) is decidable.

Solution 2. We want to construct a decider \( N \) that accepts DFAs whose language is \( \Sigma^* \), and halts and rejects on all other inputs. There are multiple possible ways to do this.

One possible program for \( N \) is the following:

On input \( \langle M \rangle \):
1. Since regular languages are closed under complement, we use \( M \) to construct a new DFA \( M' \) whose language is \( \overline{L(M)} \).
2. Let \( D \) be a decider for \( \text{E}_{\text{DFA}} \). Run \( D \) on \( M' \) to test whether \( \overline{L(M)} \) is the empty set.
3. Accept if \( D \) accepts. Reject if \( D \) rejects.

This works because \( \overline{L(M)} = \emptyset \) if and only if \( L(M) = \Sigma^* \).

Another alternative way to solve this is to define \( N \) as follows.

On input \( \langle M \rangle \):
1. Let \( A \) be a DFA whose language is known to be \( \Sigma^* \) (such as the DFA shown on the right).
2. Let \( D \) be a decider for \( \text{EQ}_{\text{DFA}} \). Run \( D \) on \( \langle M, A \rangle \) to test whether \( M \) and \( A \) recognize the same language.
3. Accept if \( D \) accepts. Reject if \( D \) rejects.

This works because \( L(M) = L(A) \) if and only if \( L(M) = \Sigma^* \).

Example 3. Every CFL is decidable.

Solution 3. Let \( A \) be a CFL. We want to create a decider \( M \) that decides \( A \).

Since \( A \) is a CFL, there exists a context-free grammar \( G \) such that \( A = L(G) \). Then we can define the decider \( M \) as follows.

On input \( \langle w \rangle \):
1. Let \( D \) be a decider for \( \text{A}_{\text{CFG}} \). Run \( D \) on \( \langle G, w \rangle \) to test whether the grammar \( G \) generates \( w \).
2. Accept if \( D \) accepts. Reject if \( D \) rejects.

We have \( L(M) = A \) because \( M \) accepts precisely the strings that the grammar \( G \) generates.

You may be wondering why we can’t just have \( M \) simulate a PDA \( P \) that recognizes the CFL \( A \). The reason is that PDAs are not guaranteed to halt, so...
this would only prove that CFLs are Turing-recognizable (but not necessarily Turing decidable).

**Example 4.** Show that

\[ \text{BAL}_{\text{DFA}} = \{ \langle M \rangle \mid M \text{ is a DFA that accepts some string } w \text{ containing equal numbers of 0s and 1s} \} \]

is decidable.

**Solution 4.** Recall that \( A = \{ w \mid w \in \{0, 1\}^* \text{ and } w \text{ contains equal numbers of 0s and 1s} \} \) is a CFL. In Week 3 recitation, we proved that the intersection of a regular language and a CFL is a CFL. So for any DFA \( M \), we know \( L(M) \cap A \) is a CFL.

Now we can construct a decider \( M \) for \( \text{BAL}_{\text{DFA}} \), as follows.

<table>
<thead>
<tr>
<th>On input ( { M } ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let ( C = L(M) \cap A ). Since ( C ) is a CFL, let ( G ) be a CFG such that ( L(G) = C ).</td>
</tr>
<tr>
<td>2. Let ( D ) be a decider for ( E_{\text{CFG}} ). Run ( D ) on ( \langle G \rangle ) to test if ( L(G) = \emptyset ).</td>
</tr>
<tr>
<td>3. <strong>Accept</strong> if ( D ) rejects. <strong>Reject</strong> if ( D ) accepts.</td>
</tr>
</tbody>
</table>

This works because \( C = L(M) \cap A \neq \emptyset \) if and only if \( L(M) \) contains some string with equal numbers of 0s and 1s.

**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 Turing-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 \( M_{\cup} \) that decides \( A \cup B \).

<table>
<thead>
<tr>
<th>On input string ( w ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Run ( M_A ) on ( w ).</td>
</tr>
<tr>
<td>2. Run ( M_B ) on ( w ).</td>
</tr>
</tbody>
</table>
3. **Accept** if either $M_A$ or $M_B$ accepts. **Reject** if both reject.

**Concatenation.** We construct a Turing decider $M_C$ that decides $A \circ B$.

On input $w$:
1. Nondeterministically guess where we split $w$ into two strings: $w = w_1w_2$.
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 $M_S$ that decides $A^*$.

On input $w$:
1. Nondeterministically guess the number $k$ of partitions. Then guess where we split $w$ into $k$ strings: $w = w_1w_2...w_k$.
2. Sequentially run $M_A$ on $w_1$, $w_2$, ..., $w_k$.
3. **Accept** if on some branch of the computation, $M_A$ accepts on all the strings. **Reject** otherwise.

**Intersection.** We construct a Turing decider $M_I$ that decides $A \cap B$.

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 $M'$ that decides $\overline{A}$.

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 $M_{U}$ 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 $w_1, w_2, \cdots, w_k$ 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 $w = w_1w_2\cdots w_k$ by looping (in that branch). This is okay because in any given branch, we want to reject if $M_A$ rejects any of $w_1, w_2, \cdots, w_k$.

**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 $M_I$ 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.

<table>
<thead>
<tr>
<th>Class of language</th>
<th>Closed under</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>union, concatenation, star, intersection, complement</td>
</tr>
<tr>
<td>Non-regular</td>
<td>complement</td>
</tr>
<tr>
<td>CFL</td>
<td>union, concatenation, star</td>
</tr>
<tr>
<td>T-recognizable</td>
<td>union, concatenation, star, intersection</td>
</tr>
<tr>
<td>T-decidable</td>
<td>union, concatenation, star, intersection, complement</td>
</tr>
</tbody>
</table>