Recitation 07: Recursion Theorem, Midterm Review

Recursion Theorem

In lecture, we talked about the idea of self-replicating Turing machines, and how this leads to the result that a Turing machine can access its own description. We did not cover the recursion theorem extensively in recitation. Many problems that can be solved using the Recursion Theorem can also be solved using general reductions, but using this theorem can often shorten/simplify the proofs. Recall that we saw an example in lecture which was an alternative proof for the undecidability of $A_{TM}$:

Another usage of the recursion theorem is for the problem of determining whether a TM is the TM with the shortest description recognizing its language.

Definition. A TM $M$ is called minimal if no TM with a shorter description can recognize $L(M)$. Let $MIN_{TM} = \{ \langle M \rangle \mid M$ is a minimal TM$\}$.

Theorem 1. $MIN_{TM}$ is not recognizable.

Proof. Recall that a language is Turing-recognizable if and only if it is enumerable. Then, assume for the sake of contradiction that there is an enumerator $E$ for $MIN_{TM}$.

We then construct the following TM $R$:

<table>
<thead>
<tr>
<th>On input $w$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Get the machine’s own description $\langle R \rangle$ via the Recursion Theorem.</td>
</tr>
<tr>
<td>2. Run $E$ until we get some $\langle B \rangle$ that is longer than $\langle R \rangle$. This is guaranteed to happen since $MIN_{TM}$ is infinite.</td>
</tr>
<tr>
<td>3. Run $B$ on $w$, and output the same thing as $B$.</td>
</tr>
</tbody>
</table>

This means that $B$ is not a minimal TM, since $R$ does the same thing as $B$ and has a shorter description. Therefore, $E$ is not an enumerator for $MIN_{TM}$ and we have reached a contradiction. □
These examples show how the recursion theorem can be used to show undecidability and unrecognizability via proof by contradiction.

**Midterm Review**

We start with the following Venn diagram of language classes, which is a useful reference to understand the relationship between regular, context-free, recognizable, and decidable languages. The diagram also contains some example languages in each class, which may be helpful when using closure properties.

Many of these languages are useful when proving a language is decidable, as well as constructing general reductions and mapping reductions, as we will see in some tips and tricks given later in this section.

Next is a table with the closure properties for each class of language. In recitation, we quickly reviewed the arguments for why these closure properties hold, and we recommend it as a review exercise that you do this yourself. This may help you find some knowledge gaps you may have with each computation model.

<table>
<thead>
<tr>
<th>Class</th>
<th>Union (∪)</th>
<th>Intersection (∩)</th>
<th>Concat (⊙)</th>
<th>Star (⁎)</th>
<th>Complement (L)</th>
<th>Reversal (L^R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CFL</td>
<td>Yes</td>
<td>No†</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>T-decidable</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>T-recognizable</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

† CFL ∩ Regular = CFL

Finally, here is a quick overview of the techniques we have studied so far to solve different kinds of problems, as well as tips and tricks for each one.
Let’s first review the general approach we take when showing a given language $L$ belongs to a certain class:

**Showing regular** The easiest solution is often to construct an NFA recognizing $L$, although closure properties can come in handy as well. The NFA allows you to use nondeterminism, which will often simplify your solution.

Conversely, if a problem tells you that a language $A$ is regular, you can take a DFA $M$ for this language instead of a Turing machine. Since DFAs have less features than TMs, we have more guarantees about $M$’s computation, which you can then use to construct another automaton.

**Showing context-free** Approaching these problems using PDAs tends to be easier than using CFGs, since PDAs allow you to think in terms of computation rather than trying to construct a grammar for $L$. The PDA’s stack is often used for counting/comparing certain amounts, and the nondeterminism can be used to make necessary guesses.

If a problem tells you that some language $A$ is a CFL, there are situations where taking the CFG $G$ for $A$ will yield a simpler solution than using a PDA $P$ that recognizes $A$.

**Showing Turing-recognizable** Here, you can construct a Turing machine $M$ and you only have to argue why it always halts for inputs $x \in L$, and that any $x$ not in $L$ is not accepted. There is no need to worry about whether $M$ halts on inputs not in the language.

**Showing decidable** One approach is to construct a Turing machine $D$ and argue why it halts on inputs that are both in the language and not in the language. As we saw in PSet 2 Problem 6, one way to guarantee this is to somehow upper-bound the number of steps for which we run $D$ on its input.

If $L$ is the language of a DFA, NFA, CFG, or PDA, it can also be useful to use a decider for languages we already proved decidable (found in the Venn Diagram above), including $A_{DFA}, E_{DFA}, EQ_{DFA}, A_{CFG}, E_{CFG}$. This will be similar to a reduction, where you modify your input to get something that can be passed into the decider for one of these languages, and it will often make it much easier to argue that your decider $D$ halts on all inputs.

We finish off this review by reviewing some tips for showing that a language $L$ **does not** belong to one of the classes we have studied:

**Showing nonregular** This is most commonly done using the Pumping Lemma, where we assume the language is regular and thus has a pumping length $p$. We then show that there is some string $s \in L$
that violates the lemma. Remember that you only need to give one string \( s \), constructed for some general \( p \), but you need to argue that there is no way to split it up into \( s = xyz \) such that \( xy^iz \in L \) for every \( i \). In other words, you have to argue that no matter how the string gets cut up, there is some \( i \) for which \( xy^iz \notin L \). Sometimes it will be easier to “pump up” \((i > 1)\) and others to “pump down” \((i = 0)\).

Sometimes, it is possible to use closure properties to show that if \( L \) is regular then some other \( L' \) (which is known to be non-regular) is also regular, which gives a contradiction.

**Showing non-CFL** The Context-Free Pumping Lemma can be used here, similarly to the Regular case. Nonetheless, it often happens that the Context-Free version requires some more case work, mainly because there are more ways to create the \( s = uvxyz \) partition. Look out for cases in which this casework can be simplified by using closure properties, or the fact that \( \text{CFL} \cap \text{REG} = \text{CFL} \). More specifically, if you want to show \( L \) is not a CFL, you can construct a language \( L' \) by taking the intersection of \( L \) with a regular language. Then you use the Pumping Lemma to show \( L' \) is not a CFL, which implies that \( L \) isn’t either.

**Showing undecidable** The approach we have used most frequently for showing undecidability is reducing \( A_{TM} \) to \( L \). We studied two main ways to do this:

- **Assume there exists a decider \( R \) for \( L \), use it to construct a decider \( S \) for \( A_{TM} \).**
- **Use a computation history strategy.** To decide if \( \langle M, w \rangle \in A_{TM} \), assume there is a decider \( R \) for \( L \), and use it to decide whether there exists an accepting computation history of \( M \) on \( w \).

It might also be easier to reduce another undecidable problem to \( L \), like \( \text{HALT}_{TM} \) or \( \text{PCP} \). We can also use diagonalization or the recursion theorem to reach a contradiction, as seen in proofs of \( A_{TM} \), but a standard reduction is often enough.

**Showing Turing-unrecognizable** The most common approach we studied was giving a mapping reduction from \( A_{TM} \) to \( L \), namely showing that \( A_{TM} \leq_m L \). It suffices to construct a computable function \( f \) such that \( x \in A_{TM} \iff f(x) \in L \).

This concludes our review of problem-solving techniques and also our midterm review. Note that reading through these is not enough practice to succeed in the exam, and we strongly recommend you try the practice problems to get a feeling of what concepts are less clear to you.