Recitation 06: Recursion Theorem + Midterm Review

Recursion Theorem

The recursion theorem is informally stated as follows.

Theorem 1 (Recursion theorem, informal). *A Turing machine can obtain a description of itself and compute on it. In other words, when writing pseudocode for a TM R, we're allowed to write "Get self-description $\langle R\rangle$" as the first step and use $\langle R\rangle$ in subsequent steps.*

The recursion theorem, and more broadly the idea of self-reference (e.g., diagonalization), is a powerful tool for proving certain things are impossible. Here's an analogy for what a proof with the recursion theorem generally looks like:

> *Suppose there's this prophet that can make certain kinds of predictions about people. I'll ask the prophet to make such a prediction about myself, and then I'll "disobey" the prediction by behaving differently from what the prophet predicted about me. This is a contradiction, and hence the prophet cannot exist.*

Now, change "prophet that makes certain kinds of predictions about people" to something like "TM D that decides whether a given TM has a certain property"[1] and the pronoun "I" to a TM R that invokes the recursion theorem. And we have a template for proofs that use the recursion theorem:[2]

[1] The prophet can also be other kinds of things that in a sense "tell you something about TMs"—see Example 2 below.

[2] Such a proof doesn't work for all TM properties. Indeed, some TM properties *are decidable*, such as the one from Recitation 3: $\{\langle M, w\rangle \mid M \text{ ever attempts to move its head left on input } w\}$. What goes wrong when you try to use the recursion theorem to show that this language is undecidable?

Suppose there's a decider D for some TM property. Construct a TM

R = "On input x,

```
1. Get self-description $\langle R\rangle$.
2. Ask $D$ whether $R$ has the property in question by running $D$ on $\langle R\rangle$.
3. Behave opposite to what $D$ said about $R$:
   • If $D$ says $R$ has the property, then don't exhibit that property.
   • If $D$ says $R$ doesn't have the property, then exhibit that property."
```

This is a contradiction: R contradicts what D says about R. Thus, the TM property in question is undecidable.

As a simple example, let's show that A_{TM} is undecidable using the recursion theorem. Note the similarity with the diagonalization proof from lecture, which also uses self-reference to construct a TM that contradicts what an A_{TM} decider says about it.

Example 1. *Show that A_{TM} is undecidable using the recursion theorem.*

Proof. Suppose for the sake of contradiction that D decides A_{TM}. Then construct TM

$R =$ "On input w,

1. Get self-description $\langle R \rangle$.
2. Run D on $\langle R, w \rangle$.
3. Behave opposite to what D said about R on w:
    - If D accepted (i.e., predicted that R would accept w), then reject.
    - If D rejected (i.e., predicted that R would reject w), then accept."

Then R accepts w iff D rejects $\langle R, w \rangle$, so D cannot be a decider for A_{TM}, a contradiction. □

While the example above identifies "prophet" with "a decider D for A_{TM}", the "prophet" doesn't always have to be a decider for a TM property. It can more generally be things that "tell you something about TMs" in an informal sense. For example, in the MIN_{TM} T-unrecognizability proof from lecture, the "prophet" is an enumerator for a TM property (namely, minimality), and the enumerator tells you something about the TMs that it enumerates (i.e., that they have that property and no other TM does). In the following example, the "prophet" is a function f from TMs to TMs, and f tells you something about the relationship between $\langle M \rangle$ and $f(\langle M \rangle)$.

Example 2. *Show that $ALL_{\mathsf{TM}} \not\leq_{\mathrm{m}} \overline{ALL_{\mathsf{TM}}}$ using the recursion theorem.*

Proof. Suppose for the sake of contradiction that a mapping f implementing the reduction exists. Then construct the TM

$R =$ "On input x,

1. Get self-description $\langle R \rangle$.
2. Get $\langle R' \rangle = f(\langle R \rangle)$.
3. Simulate R' on x."

Then f says that $L(R) \neq L(R')$, but R disobeys that prediction by behaving the same way as R' (Step 3), resulting in $L(R) = L(R')$, a contradiction. Hence, a mapping f that implements the reduction $ALL_{\mathsf{TM}} \not\leq_{\mathrm{m}} \overline{ALL_{\mathsf{TM}}}$ cannot exist. □

Midterm review

We start with a Venn diagram of language classes, which shows the relationships between regular, context-free, Turing-recognizable, co-Turing-recognizable,[3] and decidable languages (Figure 1). The diagram also contains example languages in each class. Note that the intersection of Turing-recognizable and co-Turing-recognizable languages are the decidable languages.

[3] The *co-Turing-recognizable* languages are defined to be the complements of the Turing-recognizable languages.

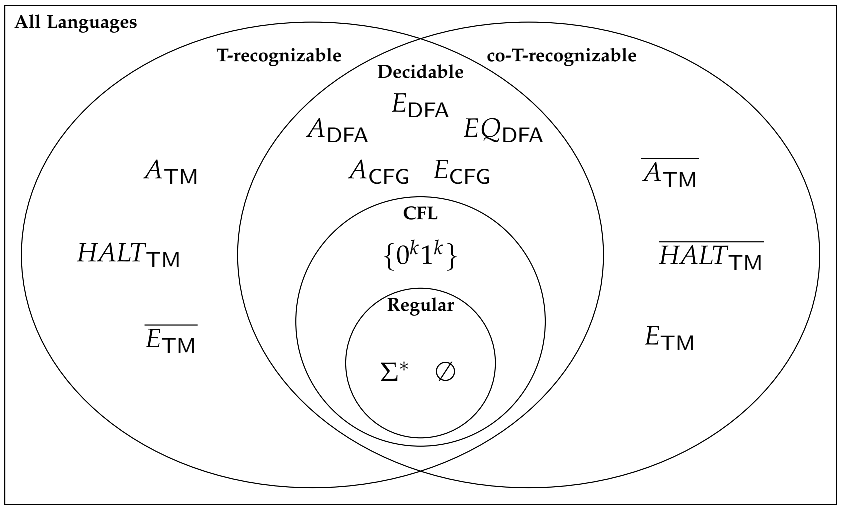

Figure 1: Venn diagram showing the language classes: regular, context-free, (co-)Turing-recognizable, decidable.

A summary of these language classes is given in Table 1.

X	What recognizes an X language	What generates an X language
Regular	DFA/NFA	Regular expression
CFL	(non-det.) PDA	CFG
T-decidable	Turing decider	—
T-recognizable	Turing machine	—

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

Next is a table with the closure properties for each language class Table 2. In recitation, we quickly reviewed the arguments for why some of these closure properties hold, and we recommend it as a review exercise that you do this yourself for all the closure properties. This may help you find some knowledge gaps you may have with each model of computation.

Observe that consistent with our diagram above, the complement of T-recognizable is not necessarily T-recognizable (otherwise, T-recognizable = co-T-recognizable, i.e., they would be represented by the same circle in the Venn diagram). This is because T-recognizers can reject an input

Class	$\cup$	$\cap$	$\circ$	$*$	$\overline{L}$	$L^{\mathcal{R}}$
Regular	Y	Y	Y	Y	Y	Y
CFL	Y	N†	Y	Y	N	Y
T-decidable	Y	Y	Y	Y	Y	Y
T-recognizable	Y	Y	Y	Y	N	Y

† CFL $\cap$ Regular = CFL

Table 2: Summary of closure properties of each language class.

by running on it forever, so the argument that we can flip the output to obtain the complement, while it works for deciders, does not work for recognizers.

The next table summarizes the decidability of questions regarding various models of computation Table 3. You can use these results to show the (un)decidability of other questions about models of computation.

X	A_X	E_X	ALL_X	EQ_X
DFA/NFA	Y	Y	Y	Y
CFG/PDA	Y	Y	N	N
LBA	Y	N	N	N
TM	N^a	N^b	N^c	N^c

^{a} T-recog. ^{b} co-T-recog. ^{c} neither T-recog. nor co-T-recog.

Table 3: Summary of decidability of questions regarding various models of computation.

We now give an overview of the techniques we have studied so far to solve different kinds of problems, as well as tips and tricks for each one. A summary is given in Table 4. All examples are from the Jeopardy I created for my recitation section: `https://jeopardylabs.com/play/what-kind-of-language`. It is recommended to try to solve the yourself before looking at the solutions.

Showing a language belongs to a certain class

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 or write a regular expression for L, although closure properties can come in handy as well. One can also construct a DFA, but an NFA allows you to use nondeterminism, which can 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. You can then, for example, construct another automaton M' out of M to prove some related

X	Show is X	Show is not X
Regular	Construct DFA/NFA/regex Closure properties	Pumping lemma Closure properties
CFL	Construct PDA/regex Closure properties	Pumping lemma Closure properties
Decidable	Construct decider Invoke known results (Table 3)	General/mapping reduction from undecidable language Computation history method Recursion theorem
T-recog.	Construct recognizer/enumerator Invoke known results (Table 3)	Mapping reduction from T-unrecognizable language

Table 4: Summary of common techniques for proving a language is (not) in a certain class.

language is regular (e.g., in proving closure properties), or use a decider that decides some property about M' to construct a decider that decides some property about M.

Example 3. *Show that $L = \{w \mid w$ represents a power of 2 in binary$\}$ is regular.*

Proof. A regular expression for L is $\texttt{10}^*$. □

Showing context-free Depending on the CFL, it may be easier to construct a PDA that recognizes it or to write a CFG that generates it. When constructing a PDA that recognizes L, the PDA's stack is often used for counting/comparing certain amounts, and the nondeterminism can be used to make necessary guesses. When constructing a CFG that generates L, it may be helpful to think about the different kinds of substructures that strings in L are composed of, which correspond to the variables in your grammar. The rules in your grammar dictate how these substructures combine to form strings in L.

If a problem tells you that some language A is a CFL, then you can take a CFG G that generates A or a PDA P that recognizes A. You can then, for example, construct another CFG G' out of G or PDA P' out of P to prove some related language is context-free (e.g., in proving closure properties). Or, you can use a decider that decides some property about G' or P' to construct a decider that decides some property about G or P (e.g., PSET 2 Q6).

Example 4. *Show that $L = \{\mathtt{a}^n\mathtt{b}^{2n} \mid n \geq 0\}$ is context-free.*

Proof. *(CFG construction)* The following CFG G generates L:

$$G: \quad S \to \mathtt{a}S\mathtt{bb} \mid \varepsilon$$

(PDA construction) The following PDA P recognizes L:

$P =$ "On input w,

1. Push two x's for every a read from the input. Go to step 2 when a b is read.
2. Pop one x for every b read from the input.
3. *Accept* if the stack is empty."

□

Showing Turing-recognizable Here, you can construct a Turing machine T that recognizes L. Since T is not required to be a decider, it only needs to halt for inputs $x \in L$. There is no need to worry about whether T rejects by halting or rejects by looping on inputs not in the language.

Example 5. *Show that $\overline{E}_{\mathsf{TM}}$ is Turing-recognizable.*

Proof. The following TM T recognizes L:

$T =$ "On input $\langle M \rangle$,

```
1. For $k = 1, 2, 3, \ldots$,
   (a) For each of the first $k$ strings in string order, simulate $M$ on
       it for $k$ steps.
   (b) If at any point $M$ accepts, then accept."
```

If $L(M) \neq \emptyset$, then there's some i such that the ith string in string order is accepted by M in n steps for some n. Then by the end of iteration $k = \max\{i, n\}$, T has accepted $\langle M \rangle$.

Conversely, if T accepts $\langle M \rangle$, then there's some k such that one of the first k strings in string order is accepted by M, so $L(M) \neq \emptyset$. □

Showing decidable This involves constructing a decider D for the language L. Make sure that it halts both on inputs that are in the language and inputs that are not in the language.

If L asks about some property about a DFA, NFA, CFG, or PDA, it is often helpful to use a decider for languages we already proved decidable, including $A_{\mathsf{DFA}}, E_{\mathsf{DFA}}, EQ_{\mathsf{DFA}}, A_{\mathsf{CFG}}, E_{\mathsf{CFG}}$ (see Figure 1, Table 3). This is a reduction, where you modify your input to get something that can be passed into the decider for one of these languages, giving a solution that is often simpler than if you wrote an algorithm from scratch.

Example 6. *Show that* $ALL_{\mathsf{DFA}} = \{\langle M\rangle \mid \textit{DFA } M \textit{ satisfies } L(M) = \Sigma^*\}$ *is decidable.*

Proof. The following TM D decides ALL_{DFA}:

$D =$ "On input $\langle M\rangle$,

1. Construct the DFA M' that recognizes the complement of $L(M)$.
2. Run the E_{DFA} decider on $\langle M'\rangle$ and return the result."

The decider works because $\langle M\rangle \in ALL_{\mathsf{DFA}}$ iff $\langle M'\rangle \in E_{\mathsf{DFA}}$. □

Showing a language does not belong to a certain class

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 non-regular This is most commonly done using the pumping lemma, where we assume the language is regular and thus have a pumping length p. We then show that there is some string $s \in L$ ($|s| \geq p$) 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 $s = xyz$ ($|xy| \leq p$, $|y| > 0$) 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 $xy^iz \notin L$. In some cases it is more convenient to "pump up" ($i > 1$) and in other cases 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.

Example 7. *Show that* $L = \{\mathtt{a}^n\mathtt{b}^{2n} \mid n \geq 0\}$ *is not regular.*

Proof. Suppose L is regular. For pumping length p, consider the string $s = \mathtt{a}^p\mathtt{b}^{2p} \in L$ ($|s| \geq p$). Any way of writing $s = xyz$ where $|xy| \leq p$ and $|y| > 0$ will have y be a non-empty substring of the $\mathtt{a}^p$ portion of s. So xz will have too few a's to be in L, violating the pumping lemma. □

Showing non-context-free The context-free pumping lemma can be used here, similarly to how you show non-regularity. However, using the context-free version often requires more case work, mainly because there are more ways to split s up into 5 parts $uvxyz$ ($|vxy| \leq p$,

$|vy| > 0$). Some ways of doing case work can be a lot more complicated than other ways, so it's recommended to spend some time thinking about what cases to have to simplify your solution.

Look out for situations where your solution can be simplified by closure properties. A particularly handy one is CFL $\cap$ regular is a CFL.

Example 8. *Show that* $L = \{0^k 1^l 0^m 1^n 0^k 1^l 0^m 1^n \mid k, l, m, n \geq 0\}$ *is not context-free.*

Proof. Suppose L is context-free. Then

$$L' := L \cap 0^* 1^* 0^* 1^* = \{0^a 1^b 0^a 1^b \mid a, b \geq 0\}$$

would be context-free, but it is not (see below), hence a contradiction. So L cannot be context-free.

To show that L' is not context-free, assume for the sake of contradiction that it is. Then for pumping length $p > 0$, consider the string $s = 0^p 1^p 0^p 1^p \in L'$ ($|s| \geq p$). Note that any way of writing $s = uvxyz$ ($|vxy| \leq p$, $|vy| > 0$) falls under one of two cases:

Case 1: Either v or y crosses a block boundary.[4] Then uv^2xy^2z won't even be of the form $0^*1^*0^*1^*$, since the 0's and 1's near the crossed block boundary will be mangled.

Case 2: Neither v nor y crosses a block boundary. Then since vxy can span at most 2 blocks, the 4 blocks in uv^2xy^2z cannot all have the same length.

In either case, $uv^2xy^2z \notin L$, thus violating the pumping lemma. $\square$

[4] Here, a *block* is defined to be a maximal substring with one distinct character, so that s consists of 4 blocks: 0^p, 1^p, 0^p, and 1^p.

Showing undecidable The standard approach for showing undecidability is a reduction from a known undecidable language such as A_{TM} to L. In other words, assuming there's a decider R for L, we can use it to construct a decider S for A_{TM}, which is a contradiction since A_{TM} is undecidable.

One particular approach to constructing the reduction is the *computation history method*. When L is of the form

$$\{\langle P \rangle \mid P \text{ is an instance of X problem and } P \text{ has a solution}\},$$

there's a mapping reduction from A_{TM} to L that converts $\langle M, w \rangle$ to a problem instance P where checking whether P has a solution is equivalent to checking whether there's an accepting computation history of M on w.

We can also use the recursion theorem in a proof by contradiction, as described earlier in the notes.

Example 9. *Show that ALL_{LBA} is undecidable.*

Proof. Reduce from A_{TM} using the computation history method. Here, a "problem instance" P is an LBA, and a "solution" is a string that the LBA doesn't accept.

Assuming R decides ALL_{LBA}, construct the decider S that decides A_{TM}:

$S =$ "On $\langle M, w\rangle$,

1. Construct the LBA B that checks that it's input is *not* an accepting computation history of M on w:

   $B =$ "On input x,

   (a) Check whether x begins with the start configuration of M on w. If not, *accept*.

   (b) Check whether x ends with an accepting configuration of M on w. If not, *accept*.

   (c) For each consecutive pair of configurations in x, check whether they violate M's transition function. This involves going back and forth between the two configurations, crossing off symbols as they get compared. (The crosses are erased afterwards.) Once a violation is found, *accept*."

2. Run R on B. If R accepts, then *reject*. If R rejects, then *accept*."

S works because $L(B) = \Sigma^*$ iff there's no accepting computation history of M on w, i.e., $\langle M, w\rangle \notin A_{\mathsf{TM}}$. □

Showing Turing-unrecognizable The standard approach is to construct a mapping reduction from a Turing-unrecognizable language such as $\overline{A_{\mathsf{TM}}}$ to L ($\overline{A_{\mathsf{TM}}} \leq_{\mathrm{m}} L$). This involves constructing a computable function f such that $x \in \overline{A_{\mathsf{TM}}} \iff f(x) \in L$.

Example 10. *Show that $ALL_{\mathsf{TM}} = \{\langle M\rangle \mid TM\ M$ is such that $L(M) = \Sigma^*\}$ is Turing-unrecognizable.*

Proof. We show $\overline{A_{\mathsf{TM}}} \leq_{\mathrm{m}} ALL_{\mathsf{TM}}$. The mapping f is defined as

$f(\langle M, w\rangle) =$ Turing machine T given by

$T =$ "On input x,

1. Interpret x as a natural number and run M on w for x steps.
2. If M hasn't accepted, *accept*. Otherwise, *reject*."

If M accepts w, then suppose it takes n steps. Then T rejects n so $\langle T\rangle \notin ALL_{\mathsf{TM}}$.

Conversely, if $\langle T\rangle \notin ALL_{\mathsf{TM}}$, then there's an x that T rejects. That means that M accepts w within x steps. □

This concludes our review of problem-solving techniques and also our midterm review. Besides reading through these notes, we recommend you practice with the sample midterms to get a sense for what the midterm will look like. We will also be holding midterm review sessions on Monday and Tuesday from 7:30pm to 9:30pm, where you can get further practice with the content.