CSCC63 – Week 3

This week: More recognizability, mapping reductions, and lots of examples.

The Class of Recognizable Languages

We finished last week by showing that some language (A_{TM}) was undecidable — there is no Turing machine (and therefore also no computer program) that can solve it in a finite amount of time for every input.

We then followed that argument up by saying that A_{TM} instead is what we call **recognizable**. We covered this section a bit quickly, though, and this is an important concept. So we should look at this idea a bit closer.

> **To recap:** *A language is recognizable iff there is a TM (a **recognizer**) that will accept exactly its yes-instances. If the input is a no-instance this TM might either reject or loop.*
>
> *In a sense, these languages are half-decidable. The recognizer we gave for A_{TM} was:*
>
> M_A = "On input $\langle M, w\rangle$:
> 1. Run M on w.
> 2. If it accepts, *accept.*
> 3. *Reject.*

There's an equivalent definition, though, to the class of recognizable languages, and this definition may be very useful for you. Remember the (somewhat trivial) statement we made last week about writing languages in set notation. A language L is decidable if we can write it in the form:

$$L = \{\langle \text{instance}\rangle \mid P(\langle \text{instance}\rangle)\}$$

where P is a decidable predicate – i.e., a predicate that can be decided using TMs.

> *E.g., if we start with the language*
>
> $$\text{ACC} = \{\langle M, w, k\rangle \mid M \text{ is a TM, } w \text{ is an input, } k \in \mathbb{N}, \text{ and } M \text{ accepts } w \text{ in} \leqslant k \text{ steps}\},$$
>
> then $\langle M, w, k\rangle \in \text{A}$, $\langle M, w, k\rangle \notin \text{A}$, and $(\langle M, w, |w|\rangle \notin \text{A}) \wedge (\langle M, w, |w|^2\rangle \in \text{A})$ are all decidable predicates.

We've said that A_{TM} is undecidable, but you can also see that we can characterize it in the following way:

$$A_{TM} = \{\langle M, w\rangle \mid \exists k, \langle M, w, k\rangle \in \text{A}\}.$$

That is, we can write it almost like we write decidable languages, but we allow ourselves to use a single "there exists" quantifier.

> **Important**: Suppose, in the example above, that M would accept w after 100 steps. Suppose further that I told you that the number of steps you'd need was ten. You'd check my answer and find that I was wrong. But that would not mean that M wouldn't ever accept — it would just mean that you need a bigger k.

We'll argue that the languages that can be written this way are exactly the recognizable languages: a language L is recognizable iff it can be written in the form:

$$L = \{\langle \text{instance} \rangle \mid \exists \langle \text{something} \rangle, P(\langle \text{instance, something} \rangle)\}$$

where P is a decision problem that can be decided using TMs.

For example, the following languages are recognizable:

A_{TM}: $\{\langle M, w \rangle \mid \exists k, \langle M, w, k \rangle \in A\}$

HALT: $\{\langle M, w \rangle \mid \exists k, \langle M, w, k \rangle \in H\}$

SQ: $\{\langle x \rangle \mid \exists y \in \mathbb{N}, x = y^2\}$

SAT: $\{\langle \phi \rangle \mid \phi$ is a Boolean formula with some truth assignment τ that sets ϕ to *true*$\}$

But wait — the last two of these problems are decidable!

As a matter of fact, every decidable problem is also recognizable: the class of decidable languages is a subclass of the recognizable languages.

You'll also note that the ⟨something⟩ that we're looking for isn't always a number. In the SAT language it was the truth assignment τ.

If you look at this characterization you can see why the class of recognizable languages might include some undecidable languages. The difficulty lies in the fact that we're trying to find the ⟨something⟩ under the "there exists" quantifier. If we can bound the search space or otherwise set things up so we can reliably find this value, the overall problem will be decidable. If, on the other hand, we have an infinite number of values to search we may find ourselves in a situation where the search will sometimes fail to finish.

Of course, it would be a bit awkward to keep calling the thing under the "there exists" quantifier the ⟨something⟩. So we'll give it a name: we'll call it a **certificate**. So the class of recognizable languages can be characterized as the L such that:

$$L = \{\langle \text{instance} \rangle \mid \exists \langle \text{certificate} \rangle, P(\langle \text{instance, certificate} \rangle)\}$$

where P is a decision problem that can be decided using TMs.

Let's see why the two characterizations of recognizable languages are equivalent:

To give a high-level argument that L is recognizable iff

$$L = \{\langle \text{instance} \rangle \mid \exists \langle \text{certificate} \rangle, P(\langle \text{instance, certificate} \rangle)\}$$

for some decidable predicate P:

- Suppose that a language L is recognized by the TM R. Then,

  $$L = \{\langle x \rangle \mid \exists k, R \text{ accepts } \langle x \rangle \text{ in } k \text{ steps}\}$$

- Suppose that $L = \{\langle \text{instance} \rangle \mid \exists \langle \text{certificate} \rangle, P(\langle \text{instance, certificate} \rangle)\}$. Then we can build a recognizer R that loops over all of the possible certificates and checks them with a decider for P.

Of course, search all of the certificates in this way we need to be sure that we can loop over said certificates. This might be tricky of they aren't numbers. So we'll need to make use of something called an **effective enumeration**.

The idea here is that we can loop over strings and TMs, as well as numbers — for example, if we have the alphabet $\Sigma = \{0, 1\}$, then we can order its strings first by length, and then by lexicographic order (i.e., in shortlex order). So the order of the strings would be:

$$\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$$

You can write a program that will output exactly this sequence, and you can use this sequence in a loop. Moreover, you can see that every finite string will eventually be reached by such a loop. This sort of ordering is called an ***effective enumeration*** *of the strings.*

Practically, once we have shown, as we have above, that we can loop over a set like the strings, we don't explicitly write out the order every time we want to use it in a loop. Instead, we start our program by saying "Let $w_0, w_1, \ldots$ be an effective enumeration of the strings."

I'll say without proof that you can effectively order the TM descriptions, as well, and you can make use this fact.

The certificate definition gives us a nice way of describing the recognizable languages, but it also tells us why, philosophically speaking, the recognizable languages are interesting.

Suppose we are looking at a language L that is recognizable but not decidable. Now, if I give you an x and claim that $\langle x \rangle \in L$, you would not not have any way of determining if I am telling the truth. But if I give you the certificate c as well, you can use the two together to check that I am telling the truth.

The decidable languages are the ones that we be **solve** in a finite amount of time, the recognizable languages are the ones where we can **check** our answers in a finite amount of time.

The difference between these ideas shows up in many places — take mathematical proofs, for example. If I give you a mathematical proof you can check in a finite amount of time whether that proof is correct. But finding that proof (i.e., solving the problem) may be very difficult.

> *You may have noticed this yourself…*
>
> *Have you ever studied for a test by going over proofs and examples from class until every step makes sense, but forgotten to ask why those steps were taken in the first place?*
>
> *If so, you'll have found the test is much harder than you'd expect given the amount of studying you've done — you'll have learned to check answers as opposed to finding them.*

Recognizability and Co-recognizability

So we've introduced the class of recognizable languages. Last week we also introduced the class of **co-recognizable** languages. We said that a language L is co-recognizable iff its compliment $\overline{L}$ is recognizable. By looking through the previous section we can see that we can characterize a language L as co-recognizable iff:

1) There is a recognizer R that recognizes accepts the language $\overline{L}$. That is, for any x:

   - If $\langle x \rangle \in \overline{L}$ (i.e., if $x \notin L$) then R accepts.
   - If $\langle x \rangle \notin \overline{L}$ (i.e., if $x \in L$) then R either rejects or loops.

1′) *If we flip the* accept *and* reject *states of R we get a TM R' for which:*

   - *If $\langle x \rangle \notin L$ then R' rejects.*
   - *If $\langle x \rangle \in \overline{L}$ then R' either accepts or loops.*

   R' recognizes a language by rejecting invalid candidates

   I'll sometimes refer to such an R' as a **co-recognizer**.

2) $\overline{L}$ can be written as

$$\overline{L} = \left\{ \langle \text{instance} \rangle \;\middle|\; \exists \langle \text{certificate} \rangle, P(\langle \text{instance, certificate} \rangle) \right\}$$

   where P is a decision problem that can be decided using TMs.

   But then, we can write L as

$$L = \left\{ \langle \text{instance} \rangle \;\middle|\; \forall \langle \text{certificate} \rangle, \neg P(\langle \text{instance, certificate} \rangle) \right\}$$

   (just take the negation of $\overline{L}$)

   We can see that if P is decidable, then so is $\neg P$: just take a decider for P and flip its answer. So if we write $Q = \neg P$, we can characterize the co-recognizable languages as the languages that can be written as

$$L = \left\{ \langle \text{instance} \rangle \;\middle|\; \forall \langle \text{certificate} \rangle, Q(\langle \text{instance, certificate} \rangle) \right\}$$

   where Q is a decision problem that can be decided using TMs.

Now, if a language is decidable, then it is also co-recognizable. A similar reasoning applies here as to the recognizable languages.

But we can do a bit better: if a language is both recognizable and co-recognizable, then it is decidable.

Why?

Well, suppose that L is both recognizable and co-recognizable. This means that there is some TM M_1 that recognizes L and another TM M_2 that recognizes $\overline{L}$.

Now, consider the following code:

M = "On input w:

1. For $k = 0, 1, 2, \ldots$:
2. Run M_1 on w for k steps.
3. If it accepts, *accept*.
4. Run M_2 on w for k steps.
5. If it accepts, *reject*.

For any input w we have two possibilities: either $w \in L$ of $w \notin L$ ($w \in \overline{L}$).

- If $\in L$ then there is some k_1 such that M_1 will accept L in k_1 steps. so M will accept in loop iteration $k = k_1$.
- If $\notin L$ then there is some k_2 such that M_2 will accept L in k_2 steps. so M will reject in loop iteration $k = k_2$.

In either case, M will eventually halt, and will accept iff $x \in L$. So M decides L, and thus L is decidable.

This approach to running many TMs — that is, running each for a set number of steps, and slowly increasing the number of steps, is called **dovetailing**. It allows us to run different TMs on different inputs without worrying about looping. It's a very good way to build recognizers for complicated languages, so we'll see more of it later (in particular, if you look ahead you'll see an example for the language the E_{TM}).

We use dovetailing when we want to do a brute-force search that requires multiple (sometimes nested) infinite loops. Think about what happens if you write multiple infinite loops:

M = "On input w:

1. For $i = 0$ to ∞:
2. For $j = 0$ to ∞:
3. $\langle$Some code$\rangle$
4. For $k = 0$ to ∞:
5. $\langle$Some more code$\rangle$

If you run this code, you'll start the i loop, start the j loop, and get stuck there forever. You'll never get to $i = 1$, and you'll certainly never get to the k loop (we're ignoring break points for the moment).

Obviously you don't want this, but you may want code that does something like the loops above. That's what dovetailing does. The idea is to have a control loop that goes to ∞, and to nest all of the other loops inside of it. All of the nested loops will iterate only to the index of the outer loop:

M = "On input w:

1. For $s = 0$ to ∞:
2. For $i = 0$ to s:
3. For $j = 0$ to s:
4. $\langle$Some code$\rangle$
5. For $k = 0$ to s:
6. $\langle$Some more code$\rangle$

This new loop structure will do what you would intuitively want the earlier loop to do, but will not run into the problem of one infinite loop blocking all of the others from running.

Now, dovetailing is very useful in conjunction with a type of machine called an **enumerator**. An enumerator is a TM that is designed to loop forever, but which has access to a special output tape. The enumerator will periodically print strings to this output tape.

Here's an example:

Let $w_0, w_1, \ldots$ be an effective ordering of the strings,
and $\langle M_0 \rangle, \langle M_1 \rangle, \ldots$ be an effective enumeration of the TMs.

M = "Ignore input:

1. For $k = 0, 1, 2, \ldots$:
2. For $i = 0$ to k:
3. For $j = 0$ to k:
4. Run M_i on w_j for k steps.
5. If it accepts, print $\langle M_i, w_j \rangle$ and continue.

You can see that this program will print exactly the strings in $\mathrm{A_{TM}}$.

You can see that in conjunction with dovetailing, this type of machine can be very powerful — we can even show that a language is recognizable iff there is some enumerator for it.

Mapping Reductions

So far, we've defined three classes of languages: decidable, recognizable, and co-recognizable. There are more, but these are a start.

We've also proven that $\mathrm{A_{TM}}$ is recognizable but not decidable (it is therefore also not co-recognizable either). The proof technique we used for that was called diagonalization. Here we'll introduce the other main proof technique we'll use in this course: the **reduction**.

Simply put, a reduction is a program – we'll assume that we have some program P_B that will solve a problem B, and we'll use P_B as a component of a larger program to solve another problem A

OK, but how do we do that?

Well, as an example: we've said that $\mathrm{A_{TM}}$ is recognizable but not decidable. We can also see that HALT is recognizable. But is it decidable?

It turns out that it isn't. Now, we could prove this by setting up another diagonalization argument, but now that we know that $\mathrm{A_{TM}}$ is not decidable we can make use of that knowledge in our proofs as well.

Suppose I have an instance $\langle M, w\rangle$ of $\mathrm{A_{TM}}$. Consider what happens if I run it through the following code:

P = "On input $\langle M, w\rangle$:

1. Let M' = "On input x:
    1. Run M on x.
    2. If it accepts, *accept*.
    3. Otherwise, *loop*."
2. Output $\langle M', w\rangle$."

I started with an instance $\langle M, w\rangle$ of $\mathrm{A_{TM}}$, but now I have another instance $\langle M', w\rangle$. Clearly, if M accepts w then so does M', and if it doesn't accept then neither does M'. But M' also has the property that if it doesn't accept, it loops forever. It never rejects.

Now, this means that M' halts on w iff M accepts w. That is, $\langle M', w\rangle \in \mathrm{HALT}$ iff $\langle M, w\rangle \in \mathrm{A_{TM}}$.

This means that HALT cannot be decidable either: if we had a TM M_H that could decide HALT, then we could use it to decide $\mathrm{A_{TM}}$ as well: given an instance $\langle M, w\rangle$, we could:

- Run it through P to get $\langle M', w\rangle$,
- run $\langle M', w\rangle$ through M_H, and then
- output the result we get from M_H.

So we've written a program P that transforms instances of $\mathrm{A_{TM}}$ into instances of HALT in such a way that we can use a solver for HALT to also solve $\mathrm{A_{TM}}$ as well.

There are many types of reductions, but the first that we'll look at is the **mapping reduction**[1]: Given any two languages A and B, we say $A \leqslant_m B$ (A mapping reduces to B) if we have some algorithm P such that:

- P halts on all inputs (remember, it's an algorithm).
- If $x \in A$, then $P(x) \in B$.
- If $x \notin A$, then $P(x) \notin B$.

[1]You'll usually find mapping reductions to be useful in this course, but there are one or two places where something called a **Turing reduction** may be useful, as well. A Turing reduction is something like the program we wrote at the end of the week 2 lecture notes – it's a program that uses a hypothetical decider for, e.g., HALT to decide another language, e.g., $\mathrm{A_{TM}}$.

In the above example, $A = \mathrm{A_{TM}}$ and $B = \mathrm{HALT}$.

We've reduced from $\mathrm{A_{TM}}$ here, but most of the reductions we'll do in the first half of the course are from HALT. This is just because the defining property of $\langle M, w\rangle \notin \mathrm{HALT}$ is that M loops on w. If we try running M on w inside of our code, the code will stall on that step iff $\langle M, w\rangle \notin \mathrm{HALT}$. This will make writing our reductions a bit easier.

If were were to set $A = \mathrm{HALT}$ (or $\overline{\mathrm{HALT}}$), and if B took instances of the form $\langle M\rangle$, we'd get something like:

DEFAULT PROOF FORMAT for proving that $\mathrm{HALT} \leqslant_m B$:

- Consider the TM P "On input $\langle M, w\rangle$ (i.e., the input to HALT):
  1. Construct the TM M_0 as follows:
     M_0 = "On input $\langle$ Input $\rangle$:
     1. $\langle$ Step 1 $\rangle$
     2. Run M on w.
     3. $\langle$ Step 3 $\rangle$"
  2. Return $\langle M_0\rangle$.
- Proof that $\langle M_0\rangle \in B$ iff $\langle M, w\rangle \in \mathrm{HALT}$.

- Note that the input to the program P is an instance of HALT (an $\langle M, w\rangle$). If we were reducing from some A other than halt, we'd use an instance of A. So if A had instances of the form, say, $\langle M\rangle$, the input to P would also be of the form $\langle M\rangle$.
- The output of P should be an instance of B. If B takes TM descriptions, this reduction would do nicely, but if its instances were of the form $\langle M, w\rangle$, you'd need to find M_0 some appropriate w, and return both. If instances of B were of the form $\langle M, N\rangle$, you'd need to find an M_0 like above, but also an N_0.

Now, if we could prove that $\langle M_0\rangle \notin B$, what would that tell us about $\langle M, w\rangle$?

It would tell us that $\langle M, w\rangle \in \overline{\mathrm{HALT}}$.

. . .In fact, we'd have a certificate (i.e., a finite proof) that $\langle M, w\rangle \in \overline{\mathrm{HALT}}$. *Since* $\langle M, w\rangle$ *is arbitrary we'd always be ble to prove when an* $\langle M, w\rangle$ *is looping. This would mean that* $\overline{HALT}$ *would be recognizable.*

But then HALT *would be both recognizable and co-recognizable (and hence decidable), and we know that can't be true.*

So since we know that $\overline{\mathrm{HALT}}$ is not recognizable, we would have just proven that B is not recognizable (alternatively, that $\overline{B}$ is not co-recongizable).

In fact, from the text, we find:

Theorem 5.22:

If $A \leqslant_m B$ and B is decidable, then A is decidable.

Corollary 5.23:

If $A \leqslant_m B$ and A is undecidable, then B is undecidable.

Theorem 5.28:

If $A \leqslant_m B$ and B is recognizable, then A is recognizable.

Corollary 5.29:

If $A \leqslant_m B$ and A is not recognizable, then neither is B.

In addition, if we know that $A \leqslant_m B$, then what can we say about $\overline{A}$ and $\overline{B}$?

We know that $\overline{A} \leqslant_m \overline{B}$. Can you see why?

Some Examples:

OK, that's the theory, so let's look at a few languages to get a feel for how this all works.

$$\mathrm{E}_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \varnothing\}$$

Note: $L(M)$ is the set of strings that M accepts, i.e., $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$

$$L_1 = \{\langle M \rangle \mid \exists \text{ a TM } N \text{ such that } M \text{ accepts } \langle N \rangle \text{ and } N \text{ accepts } \varepsilon\}$$

$$L_2 = \{\langle M \rangle \mid L(M) = \{1\}\}.$$

$$\mathrm{ALL}_{TM} = \{\langle M \rangle \mid M \text{ accepts every input}\}$$

We'll determine now whether these languages are recognizable, co-recognizable, both, or neither.

Let's do this by looking at each language, and asking whether there's an extra piece of information — a certificate — that would make the whole thing decidable.

To start off: if you want to convince me that some $\langle M \rangle$ is *not* in $\mathrm{E_{TM}}$, what sort of information could you give me?

> $\langle M \rangle \in \mathrm{E_{TM}}$ iff M does not accept any input. So I might try to convince you that it accepts some input by telling you an input w that it accepts.
>
> *So you take the w, run M on it, and it starts running. And running. And running…Is there any point that you could be sure that I had given you a bad input, one that makes M loop? Is there any point where you could be sure that there's no way it would work if just ran it a little longer?*
>
> No, there's not — we know that since we know that $\mathrm{A_{TM}}$ is undecidable. So we need just a bit more information.
>
> Suppose that instead I gave you an input w and a time k, and stated that M accepts w within k steps?
>
> *You can check this!*
>
> So E_{TM} is co-recognizable.

Can you think of any way you might be able to prove that $\langle M \rangle$ is in $\mathrm{E_{TM}}$?

> Probably not. We'll prove this later, but the answer is no. Intuitively, I can't prove to you that M fails to accept any *one* input w. How could I possibly do so for *every* input?
>
> So $\mathrm{E_{TM}}$ is likely not recognizable.

What about L_1? How would you prove that an $\langle M \rangle$ is in L_1? Not in L_1?

> **To show an $\langle M \rangle$ is in L_1:** If you wanted to prove to me that $\langle M \rangle \in L_1$, you could give be the $\langle N \rangle$, along with the time needed to run M and N.
>
> **To show an $\langle M \rangle$ is not in L_1:** You probbly can't — if M loops on everything, then it would not be in L_1, but we wouldn't generally be able to prove that it was looping.

So we guess that L_1 is recognizable but not co-recognizable.

Can you see any easy certificates to prove that a given $\langle M \rangle$ is or is not in L_2? Why or why not?

> **It's probably not recognizable** — if it accepted 1 and looped on everything else, how would we ensure that it was indeed looping?
>
> **It's probably not co-recognizable** — If it rejects everything other than 1 but loops on 1, how would we prove it was looping?

So we guess that L_2 is neither recognizable nor co-recognizable.

What about $\mathrm{ALL_{TM}}$? How would you prove that an $\langle M \rangle$ is in $\mathrm{ALL_{TM}}$? Not in $\mathrm{ALL_{TM}}$?

It's probably not recognizable — we can prove that an $\langle M \rangle$ halts on one input, but what sort of certificate would we need to prove it halts on all of them?

It's probably not co-recognizable — if an $\langle M \rangle$ loops on some input it is not in $\mathrm{ALL}_{\mathrm{TM}}$, but how would we prove it?

So we'll guess that $\mathrm{ALL}_{\mathrm{TM}}$ is neither recognizable nor co-recognizable.

So we've made our guesses. Now, we just need to prove them. If we've given a certificate, we've basically proven recognizability or co-recognizability, but writing a recognizer isn't too hard — it's just a matter of searching for an appropriate certificate:

Proof that E_{TM} is co-recognizable:

We'll just use dovetailing.

Let $w_0, w_1, \ldots$ be an effective enumeration of Σ^*.

Let $M_E =$ "On input $\langle M \rangle$:

1. For $k = 0$ to ∞:
2. For $i = 0$ to k:
3. Run M on w_i for k steps.
4. If it accepts, *accept*.

If you want to show that this program is a recognizer for $\overline{\mathrm{E}_{\mathrm{TM}}}$, you'll argue that:

1. If an $\langle M \rangle$ is in $\overline{\mathrm{E}_{\mathrm{TM}}}$, then M_E accepts.
2. If an $\langle M \rangle$ is not in $\overline{\mathrm{E}_{\mathrm{TM}}}$, then M_E does not accept.

I'll leave the details as an exercise for the reader.

But now we want to prove that E_{TM} is not recognizable (that $\overline{\mathrm{E}_{\mathrm{TM}}}$ is not co-recognizable). We'll do this by using the mapping reductions we introduced earlier.

Note that $\overline{\mathrm{E}_{\mathrm{TM}}} = \left\{ \langle M \rangle \mid \exists w \in \Sigma^*, M \text{ accepts } w \right\}$.

Here's the mapping reduction:

Proof that $\mathrm{HALT} \leqslant_m \overline{\mathrm{E}_{\mathrm{TM}}}$:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input x:
     1. Pass.
     2. Run M on w.
     3. Accept."
  2. Return $\langle M_0 \rangle$."

- We're now going to prove that $\langle M_0 \rangle \in \overline{\mathrm{E}_{\mathrm{TM}}}$ iff $\langle M, w \rangle \in \mathrm{HALT}$.

($\Leftarrow$) Suppose that $\langle M, w\rangle \in \text{HALT}$.

Then, for every input x, M_0 reaches line 2.
When it does, M will eventually halt and M_0 will reach line 3 and accept.
So M_0 will accept every input (and therefore will accept at least one input).
$\Rightarrow \langle M_0\rangle \in \overline{\text{E}_{\text{TM}}}$.

($\Rightarrow$) Suppose that $\langle M, w\rangle \notin \text{HALT}$.

Then, for every input x, M_0 reaches line 2.
When it does, M will loop, and so M_0 will never reach line 3 and accept.
So M_0 will not accept any input.
$\Rightarrow \langle M_0\rangle \in \text{E}_{\text{TM}}$.

And that's a mapping reduction — we've proven that E_{TM} is co-recognizable but not recognizable.

> Think about what we've done here. We've taken one type of problem (HALT), and embedded it in another type of problem $\overline{\text{E}_{\text{TM}}}$ — we've showed that there's a part of the instance space for $\overline{\text{E}_{\text{TM}}}$. that looks exactly like HALT. And so, we also know that $\overline{\text{E}_{\text{TM}}}$ is at least as hard to solve as HALT.

This is the sort of proof we'll usually be doing in this course to show undecidability, unrecognizability, and un-co-recognizability — the idea is we transform instances of a language A into instances of a language B. If we do this, we can use B to either decide or recognize A.

Let's see a few more examples:

Firstly, let's look at L_1:

Proof that L_1 is recognizable:

We'll use dovetailing.

> Let $\langle N_0\rangle, \langle N_1\rangle, \ldots$ be an effective enumeration of the TMs.
> Let M_{L_1} = "On input $\langle M\rangle$:
>
> 1. For $k = 0$ to ∞:
> 2. For $i = 0$ to k:
> 3. Run M on N_i for k steps.
> 4. Run N_i on ε for k steps.
> 5. If both accept within k steps, *accept*.

Note that we use the same k for both M and N. We can do this since we ask whether M and N accept *within* k steps. If M accepts in k_1 steps and N accepts within k_2 steps, then M_{L_1} will accept once $k = \max(k_1, k_2)$.

Once again, you should be prepared to justify this code.

Proof that L_1 is not co-recognizable:

Let's recap: what we want to do here is take instances $\langle M, w \rangle$ of HALT and use them to create instances $\langle M_0 \rangle$ of L_1 in such a way that $\langle M_0 \rangle \in L_1$ iff $\langle M, w \rangle \in \text{HALT}$.

Last time we had $\langle M_0 \rangle \in \text{E}_{\text{TM}}$ iff $\langle M, w \rangle \notin \text{HALT}$. Can you see why it's different this time?

As in the default reduction given above, we'll want our M_0 to have the form:

M_0 = "On input ⟨ Input ⟩:

1. ⟨ Step 1 ⟩
2. Run M on w.
3. ⟨ Step 3 ⟩"

This is for reductions from HALT. You can use it to get started, but you'll want to think about how to do reductions from other languages, too.

What this template does is it creates a program that uses its step 2 to *filter out* some of its steps:

- Whatever is in line 1 will always run.
- Whatever is on line 3 will only run if M halts on w.

As we apply this to L_1, we'll want to make sure that if M loops on w, L_1 does not accept any TM $\langle N \rangle$ such that N accepts ε. So we want to ensure that line 1 does not accept any such $\langle N \rangle$.

What can we make line 1 do to ensure it doesn't accept any $\langle N \rangle$?

Just have it pass — then it won't accept anything.

We'll also want to ensure that M_0 does accept an $\langle N \rangle$. This is what line 3 will do.

How can we make sure that line 3 accepts an appropriate $\langle N \rangle$?

Have it accept everything.

So this means we get the TM:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input x:
     1. Pass.
     2. Run M on w.
     3. Accept."
  2. Return $\langle M_0 \rangle$."

Now we need our iff proof.

Proof that $\langle M_0 \rangle \in L_1$ iff $\langle M, w \rangle \in \text{HALT}$:

($\Leftarrow$) Suppose that $\langle M, w \rangle \in \text{HALT}$.

Then, for every input x, M_0 reaches line 2.
When it does, M will eventually halt and M_0 will reach line 3 and accept.
So M_0 will accept every input.
$\Rightarrow$ M will accept $\langle N \rangle$, where N is the TM:

"On input x: 1. *accept*."

$\Rightarrow$ M accepts an $\langle N \rangle$ such that N accepts ε.
$\Rightarrow \langle M_0 \rangle \in L_1$.

($\Rightarrow$) Suppose that $\langle M, w \rangle \notin \text{HALT}$.

Then, for every input x, M_0 reaches line 2.
When it does, M will loop, and so M_0 will never reach line 3 and accept.
So M_0 will not accept any input.
$\Rightarrow \langle M_0 \rangle \notin L_1$.

Now let's have a look at L_2.

Proof that L_2 is not co-recognizable:

We'll start with co-recognizability here because it's easier (in this case).

We want to show L_2 is not co-recognizable, and so we need to show there is no general proof that an $\langle M \rangle$ is *not* in it.

This means that we need to reduce from HALT.

Again, in our default reduction — we have an instance $\langle M, w \rangle$ of HALT and want to produce an instance $\langle M_0 \rangle$ of L_2, where M_0 has the form:

M_0 = "On input $\langle$ Input $\rangle$:

1. $\langle$ Step 1 $\rangle$
2. Run M on w.
3. $\langle$ Step 3 $\rangle$"

Now, we want $L(M_0)$ to be $\{1\}$ iff M halts on w — if M loops on w we should get something else.

We want to set things up so that if M halts on w we accept 1 and nothing else, but if it loops we either don't accept 1 or we accept things other than 1.

But if M loops it blocks further computation. It's easiest to use it to prevent us from accepting things in step 3. So we'll try *not* accepting 1 if M loops.

So what should step 1 say?

> *Pass.*

On the other hand, if M halts on w, we run step 3. What can we do in step 3 to make $L(M_0) = \{1\}$?

> *Accept iff* w *is* 1.

So if we put this all together, we get the following reduction:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input x:
     1. Pass.
     2. Run M on w.
     3. If $x == 1$, *accept*, otherwise *reject*."
  2. Return $\langle M_0 \rangle$."

Proof that $\langle M_0 \rangle \in L_2$ iff $\langle M, w \rangle \in \text{HALT}$:

> ($\Leftarrow$) Suppose that $\langle M, w \rangle \in \text{HALT}$.
>
> Then, for every input x, M_0 reaches line 2.
> When it does, M will eventually halt and M_0 will reach line 3.
> When it does, it will accept iff x is 1.
> $\Rightarrow L(M_0) = \{1\}$.
> $\Rightarrow \langle M_0 \rangle \in L_2$.
>
> ($\Rightarrow$) Suppose that $\langle M, w \rangle \notin \text{HALT}$.
>
> Then, for every input x, M_0 reaches line 2.
> When it does, M will loop, and so M_0 will never reach line 3 and accept.
> So M_0 will not accept any input.
> $\Rightarrow L(M_0) \neq \{1\}$.
> $\Rightarrow \langle M_0 \rangle \notin L_2$.

Proof that L_2 is/is not recognizable:

We want to show L_2 is not recognizable, and so we need to show there is no general proof that an $\langle M \rangle$ is in it.

This means that we need to reduce from $\overline{\text{HALT}}$.

Let's go back to our default reduction — we have an instance $\langle M, w \rangle$ of $\overline{\text{HALT}}$ and want to produce an instance $\langle M_0 \rangle$ of L_2, where M_0 has the form:

M_0 = "On input $\langle$ Input $\rangle$:

1. $\langle$ Step 1 $\rangle$
2. Run M on w.
3. $\langle$ Step 3 $\rangle$"

Now, we want $L(M_0)$ to be $\{1\}$ iff M loops on w — so if M loops on w we want M_0 to accept an input of 1, and nothing else.

To do this, we'll make use of step 1. Think about the structure of this reduction. We always run the first part (step 1), but we only get to the last part (step 3) if M halts. So we're using M as a *filter*: if M loops, we don't get to step 3, and so any string we would accept there is not accepted.

- If M loops on w, $L(M_0)$ is the set of strings we accept in step 1.
- If M halts on w, $L(M_0)$ is the set of strings we accept in either step 1 or step 3.

So what should step 1 say?

> *Accept iff* x *is* 1.

Now, if M halts on w, we run step 3. What can we do in step 3 to make $L(M_0) \neq \{1\}$?

> *Accept everything else.*

So if we put this all together, we get the following reduction:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input x:
     1. If $x == 1$, *accept*.
     2. Run M on w.
     3. *Accept*."
  2. Return $\langle M_0 \rangle$."

Proof that $\langle M_0 \rangle \in L_2$ **iff** $\langle M, w \rangle \in \overline{\text{HALT}}$:

> ($\Leftarrow$) Suppose that $\langle M, w \rangle \in \overline{\text{HALT}}$.
>
> Then M_0 will accept if x is 1, otherwise it will go on to line 2.
> When it does, M will loop, and so M_0 will never reach line 3.
> So M_0 accepts only the input 1.
> $\Rightarrow L(M_0) = \{1\}$.
> $\Rightarrow \langle M_0 \rangle \in L_2$.
>
> ($\Rightarrow$) Suppose that $\langle M, w \rangle \notin \overline{\text{HALT}}$.
>
> Then M_0 will accept if x is 1, otherwise it will go on to line 2.
> When it does, M will halt, and so M_0 will reach line 3.
> Upon reaching line 3, M_0 will accept.
> So M_0 accepts every input.
> $\Rightarrow L(M_0) \neq \{1\}$.
> $\Rightarrow \langle M_0 \rangle \notin L_2$.

And so we've proven that L_2 is neither recognizable nor co-recognizable.

Now, let's do ALL_{TM}. This one will be a bit more tricky. The co-recognizability part is simpler (again, just in this case), so let's do that first.

Proof that ALL_{TM} is not co-recognizable:

We want to show ALL_{TM} is not recognizable, and so we need to show there is no general proof that an $\langle M \rangle$ is not in it.

This means that we need to reduce from HALT.

Let's go back to our default reduction — we have an instance $\langle M, w \rangle$ of HALT and want to produce an instance $\langle M_0 \rangle$ of ALL_{TM}, where M_0 of the usual form.

Now, we want M_0 to halt on every input iff M halts on w — so if M loops in step 2, there should be at least one input that it doesn't accept, and in step 3 it should accept everything.

So what should step 1 say?

> *Anything is OK, as long as it doesn't reject or loop. Let's make it pass.*

Now, if M halts on w, we run step 3. What should step 3 say?

> *We want it to accept everything. So We'll have it accept.*

So if we put this all together, we get the following reduction:

So if we put this all together, we get the following reduction:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input x:
     1. Pass.
     2. Run M on w.
     3. *Accept.*"
  2. Return $\langle M_0 \rangle$."

Proof that $\langle M_0 \rangle \in \text{ALL}_{\text{TM}}$ iff $\langle M, w \rangle \in$ HALT:

($\Leftarrow$) Suppose that $\langle M, w \rangle \in$ HALT.

Then M_0 always gets to line 2.
When it does, M will halt, and so M_0 will reach line 3 and accept.
So M_0 accepts every input.
$\Rightarrow \langle M_0 \rangle \in \text{ALL}_{\text{TM}}$.

($\Rightarrow$) Suppose that $\langle M, w\rangle \notin \text{HALT}$.

Then M_0 always gets to line 2.
When it does, M will loop, and so M_0 will never reach line 3.
So M_0 doesn't accept any input (and so doesn't accept every input, either).
$\Rightarrow \langle M_0\rangle \notin \text{ALL}_{\text{TM}}$.

Now comes the hard part:

Proof that ALL_{TM} is not recognizable:

Can you see why this is harder?

We want to show ALL_{TM} is not recognizable, and so we need to show there is no general proof that an $\langle M\rangle$ is in it.

This means that we need to reduce from $\overline{\text{HALT}}$.

Again, in our default reduction — we have an instance $\langle M, w\rangle$ of $\overline{\text{HALT}}$ and want to produce an instance $\langle M_0\rangle$ of ALL_{TM}, where M_0 has the form:

M_0 = "On input $\langle$ Input $\rangle$:

1. $\langle$ Step 1 $\rangle$
2. Run M on w.
3. $\langle$ Step 3 $\rangle$"

Now, here's the problem: we've already said that step 2 is a filter — if M loops we never reach step 3, and so anything we would otherwise accept in step 3 will not be accepted.

But now we want to *add* new strings to $L(M_0)$ if M loops — so filtering them out won't help us!

Here's how we'll get around that problem: think back to the certificates for A_{TM} or HALT. If M loops, how many of those certificates will work? If it accepts, how many will work?

So we'll treat the input to M_0 as a certificate for M halting on w. If the certificate is good, we loop. Otherwise we accept. Checking the certificate is going to mean changing step 2 of the default M_0.

So now we get the reduction:

- Consider the TM P "On input $\langle M, w\rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input $\langle k\rangle$:
     1. Pass.
     2. Run M on w for k steps.
     3. If it halts, *loop*, otherwise *accept*."
  2. Return $\langle M_0\rangle$."

In our default reduction we were running M on w as type of filter. This time we're spreading an infinite number of checks – one per k – across the space of inputs.

Proof that $\langle M_0 \rangle \in \mathrm{ALL}_{\mathrm{TM}}$ **iff** $\langle M, w \rangle \in \overline{\mathrm{HALT}}$:

($\Leftarrow$) Suppose that $\langle M, w \rangle \in \overline{\mathrm{HALT}}$.

Then M_0 always gets to line 2.
When it does, M not halt within k steps for any input k, since it loops.
So M_0 accepts every input.
$\Rightarrow \langle M_0 \rangle \in \mathrm{ALL}_{\mathrm{TM}}$.

($\Rightarrow$) Suppose that $\langle M, w \rangle \notin \overline{\mathrm{HALT}}$.

Then M_0 always gets to line 2.
When it does, M will halt for some input k.
For this k, M_0 will loop, and so it will not accept.
$\Rightarrow \langle M_0 \rangle \notin \mathrm{ALL}_{\mathrm{TM}}$.

And now we've proven that $\mathrm{ALL}_{\mathrm{TM}}$ is neither recognizable nor co-recognizable.

Practice (if there's time):

So far all of our reductions have been from HALT (or $\overline{\mathrm{HALT}}$). But let's see what happens if we don't reduce from HALT? E.g., can we show $L_2 \leqslant_m \mathrm{ALL}_{\mathrm{TM}}$?

If we do this what will our inputs and outputs be?

- Our input will be an instance of L_2, which will be a TM description $\langle M \rangle$.
- Our output will be an instance of $\mathrm{ALL}_{\mathrm{TM}}$, which will be a TM description of the form $\langle M_0 \rangle$.

Now, given an instance of L_2, how might we try to come up with an instance of $\mathrm{ALL}_{\mathrm{TM}}$?

Idea: We'll actually do a mash-up of the previous $\mathrm{ALL}_{\mathrm{TM}}$ reductions.

We want an M_0 that accepts every input iff M accepts only the string 1. So we can run M on 1, and that'll be a filter like the usual step 2s in our previous reductions.

If M accepts 1, we'll dovetail over all of its other inputs using an idea like the one we used in our previous reduction.

If we do this we get the reduction:

Let $w_0, w_1, \ldots$ be an effective enumeration of Σ^*.

- Consider the TM P "On input $\langle M \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input $\langle k \rangle$:
     1. Run M on 1.
     2. If it rejects, *loop*. Otherwise,
     3. For $i = 0$ to k:
     4. If $w_i \neq 1$:
     5. Run M on w_i for k steps.
     6. If it accepts, *loop*.
     7. *Accept*. "
  2. Return $\langle M_0 \rangle$."

Proof that $\langle M_0 \rangle \in \mathrm{ALL}_{\mathrm{TM}}$ **iff** $\langle M \rangle \in L_2$:

($\Leftarrow$) Suppose that $\langle M \rangle \in L_2$.

Then M accepts only the input 1.
So M will accept 1 in step 1, and will get to step 2 and 3.
When it does, M not accept within k steps for any input i from w_0 to w_k, since M only accepts 1.
So M_0 accepts every input.
$\Rightarrow \langle M_0 \rangle \in \mathrm{ALL}_{\mathrm{TM}}$.

($\Rightarrow$) Suppose that $\langle M \rangle \notin L_2$.

Then M either does not accept 1 or it accepts some other string.
If M does not accept 1, it will loop on either line 1 or 2.
If M does accept 1, then there is some string $w_i \neq 1$ that it also accepts.
Let k_i be the number of steps it takes to do so.
On input $\langle k \rangle$, where $k = \max(i, k_i)$, M will loop.
$\Rightarrow M_0$ will not accept this k.
$\Rightarrow \langle M_0 \rangle \notin \mathrm{ALL}_{\mathrm{TM}}$.

Finally, what does this reduction tell us?

If we know that L_2 is neither recognizable nor co-recognizable, we also know that $\mathrm{ALL}_{\mathrm{TM}}$ is neither recognizable nor co-recognizable.

The Take-Away from this Lesson:

- We've returned to the idea of recognizability and given an alternative way of looking at it.
- We've introduced mapping reductions.

Glossary:

Certificate, $L(M)$, *Mapping Reduction*

***Languages*:**

ALL_{TM}, E_{TM}, L_1, L_2