CSCC63 – Week 4

This week: More examples.

Recognizability and Co-recognizability, Continued...

Last week we started going over examples of mapping reductions to prove non-recognizability. Let's finish with ALL_{TM} nd go over a few more examples this week.

Firstly, 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 $\langle M_0 \rangle \in \mathrm{ALL}_{\mathrm{TM}}$ iff $\langle M, w \rangle \in \mathrm{HALT}$:

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

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.
$\Rightarrow \langle M_0 \rangle \in \mathrm{ALL}_{\mathrm{TM}}$.

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

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).
$\Rightarrow \langle M_0 \rangle \notin \mathrm{ALL}_{\mathrm{TM}}$.

Now comes the hard part:

Proof that $\mathrm{ALL}_{\mathrm{TM}}$ is not recognizable:

Can you see why this is harder?

We want to show $\mathrm{ALL}_{\mathrm{TM}}$ 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 $\overline{\mathrm{HALT}}$.

Again, in our default reduction — we have an instance $\langle M, w \rangle$ of $\overline{\mathrm{HALT}}$ and want to produce an instance $\langle M_0 \rangle$ of $\mathrm{ALL}_{\mathrm{TM}}$, 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.

But we can sometimes do bit better: , let's show that $L_2 \leqslant_m \mathrm{ALL}_{\mathrm{TM}}$:

1. If we do this reduction, what will our input be?

   An instance of L_2, i.e., a TM description $\langle M \rangle$.

2. What will our output be?

   An instance of $\mathrm{ALL}_{\mathrm{TM}}$, i.e., a TM description $\langle M_0 \rangle$.

Since we're not using the input $\langle M, w \rangle$ we can't do the default reduction: line 2 won't have any meaning without giving a w to run.

What we want to do is output a TM M_0 that halts on every input iff the original M accepts only the input 1 — that is, if:

- If M sees an input 1 it accepts, and
- If M sees any other input it doesn't accept.

This means that we want M_0 to do two things:

- We want it to fail to accept every input if M doesn't accept 1, and
- We want it to fail to accept some inputs if M accepts anything other than 1, it won't accept.

How could we program a TM to do these two things?

- Start by running M on 1 — if it loops M_0 will get stuck there and won't accept anything.
- Use the idea from $\mathrm{ALL}_{\mathrm{TM}}$ earlier and dovetail over all inputs other than 1 for the TM M.

If we put all of this together, 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 M_0 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, M_0 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.
$\Rightarrow \langle M_0 \rangle \notin \mathrm{ALL}_{\mathrm{TM}}$.

OK, so we've proven that $L_2 \leqslant_m \text{ALL}_{\text{TM}}$. But what does this mean?

> We know that L_2 is neither recognizable nor co-recognizable. So this shows that ALL_{TM} is also neither recognizable nor co-recognizable.
>
> *We showed this directly last week, but this is another way of showing it. Sometimes this approach is easier, since it only requires one reduction.*

More Examples:

Let's look at a few new languages:

- $\text{EQ}_{\text{TM}} = \{\langle M, N\rangle \mid L(M) = L(N)\}$
  I.e.,:

$$\text{EQ}_{\text{TM}} = \left\{ \langle M, N\rangle \;\middle|\; \begin{array}{r} \forall w, k, \exists k', (\langle M, w, k\rangle \in \text{ACC} \rightarrow \langle N, w, k'\rangle \in \text{ACC}) \\ \wedge(\langle N, w, k\rangle \in \text{ACC} \rightarrow \langle M, w, k'\rangle \in \text{ACC}) \end{array} \right\}.$$

- $L_3 = \{\langle M\rangle \mid \exists y, M \text{ outputs } y + 5 \text{ on the input } \langle y\rangle\}$
  I.e.,:

$$L_3 = \{\langle M\rangle \mid \exists y, k, M \text{outputs } y + 5 \textit{within } k \textit{ steps on the input} \langle y\rangle\}.$$

- $\text{DEC} = \{\langle M\rangle \mid L(M) \text{ is decidable}\}$
  I.e.,:

$$\text{DEC} = \left\{ \langle M\rangle \;\middle|\; \begin{array}{r} \exists N, \forall w, k, \exists k', (\langle N, w, k'\rangle \in \text{H}) \\ \wedge(\langle M, w, k\rangle \in \text{ACC} \rightarrow \langle N, w, k'\rangle \in \text{ACC}) \\ \wedge(\langle N, w, k\rangle \in \text{ACC} \rightarrow \langle M, w, k'\rangle \in \text{ACC}) \end{array} \right\}.$$

Let's start analyzing these languages with EQ_{TM}.
Firstly, do we think that EQ_{TM} is recognizable, co-recognizable, both, or neither? Why?

> *If M rejects an x and N loops on it, how could we prove it? It can't be recognizable.*
>
> *Similarly, if M accepts an x and N loops on it, how would we prove it? It can't be co-recognizable either.*
>
> As a matter of fact, EQ_{TM} is computationally equivalent to ALL_{TM}. They reduce to each other.

OK, with that done, let's get through the reductions.

The difference between these reductions and the ones we did last week will be in the output format of our reductions. Remember that every language we looked at last week had instances of the form $\langle M\rangle$, where M was a TM.

So every reduction P that we did last week had an output of the form $\langle M_0\rangle$.

But this time, EQ_{TM} has instances of the form $\langle M, N\rangle$ — so we need to output a *pair* of TMs.

Here Are the reductions:

- **Proof that EQ_{TM} is not recognizable**: $\overline{\text{HALT}} \leqslant_m \text{EQ}_{\text{TM}}$:

  We'll use something like the default reduction again, but before we do the proof, we'll look at what we want to do. Remember, we take as input an instance $\langle M, w \rangle$ of $\overline{\text{HALT}}$. We want to output a pair of TMs M_0 and N_0 so that $L(M) = L(N)$ iff M loops on w.

  If we look back at our original default reduction, we had an M_0 defined as:

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

  1. Pass.
  2. Run M on w.
  3. *Accept.*"

  What is the language of M_0 if M loops on w? If it halts?

  If M halts then M_0 accepts Σ^*.

  If it loops M_0 accepts $\varnothing$.

  With that in mind, here's the idea we'll use. We'll use this M_0, and set N_0 to be the simplest TM we can build whose language matches $L(M_0)$ when M loops on w. With this in mind, we get this 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. Construct the TM N_0 as follows:
       N_0 = "On input x:
       1. *Reject.* "
    3. Return $\langle M_0, N_0 \rangle$."

  Proof that $\langle M_0, N_0 \rangle \in \text{EQ}_{\text{TM}}$ iff $\langle M, w \rangle \in \overline{\text{HALT}}$:

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

  Then M loops on w.
  Then M_0 will loop on step 2 and will not reach step 3.
  So $L(M_0) = \varnothing = L(N_0)$.
  $\Rightarrow \langle M_0, N_0 \rangle \in \text{EQ}_{\text{TM}}$.

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

Then M halts on w.
So M_0 will complete step 2 and get to step 3.
When it does, it will accept.
So $L(M_0) = \Sigma^* \neq L(N_0)$.
$\Rightarrow \langle M_0, N_0\rangle \notin \text{EQ}_{\text{TM}}$.

- **Proof that EQ_{TM} is not co-recognizable**: $\text{HALT} \leqslant_m \text{EQ}_{\text{TM}}$:

  The idea here is actually quite similar — we have the same inputs and outputs, and we can even use the same M_0. The only difference is that we'll set N_0 to be the simplest TM we can build whose language matches $L(M_0)$ when M halts on w. *(Sometimes we can re-use our work, and it's good to be lazy when we can)* With this in mind, we get this 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. Construct the TM N_0 as follows:

     N_0 = "On input x:
     1. *Accept.* "

  3. Return $\langle M_0, N_0\rangle$."

Proof that $\langle M_0, N_0\rangle \in \text{EQ}_{\text{TM}}$ iff $\langle M, w\rangle \in \text{HALT}$:

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

Then M halts on w.
So M_0 will complete step 2 and get to step 3.
When it does, it will accept.
So $L(M_0) = \Sigma^* = L(N_0)$.
$\Rightarrow \langle M_0, N_0\rangle \in \text{EQ}_{\text{TM}}$.

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

Then M loops on w.
Then M_0 will loop on step 2 and will not reach step 3.
So $L(M_0) = \varnothing \neq L(N_0)$.
$\Rightarrow \langle M_0, N_0\rangle \notin \text{EQ}_{\text{TM}}$.

Let's have a look at the other two languages now:

Firstly, Let's go over L_3.

Do we think that L_3 is recognizable, co-recognizable, both, or neither? Why?

Recognizable but not co-recognizable.

Showing that L_3 is recognizable is actually pretty easy — we can just give a certificate for it. The certificate we'll give will be:

> $\langle y, k \rangle$, where y is the input that results in the output $y + 5$, and k is the time M takes to halt. We can build a recognizer by enumerating these certificates and checking them one by one.
>
> *I'll leave the construction of the recognizer as an exercise for the student.*

OK, with that done, let's get through the reduction to show non-co-recognizability.

This reduction is actually fairly straightforward. The reason we're including it is really so that we have an example of a reduction involving a TM that outputs a value, instead of just accepting or rejecting.

We'll use something like the default reduction again, but this time we'll change our M_0 so that it outputs a value:

M_0 = "On input $\langle$ Input $\rangle$:
1.
2. Run M on w.
3.

With this in mind, we get this reduction:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:

     M_0 = "On input $\langle y \rangle$:
     1. Pass
     2. Run M on w.
     3. Output $y + 5$. "
  2. Return $\langle M_0 \rangle$."

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

> ($\Leftarrow$) Suppose that $\langle M, w \rangle \in \text{HALT}$.
>
> Then M halts on w.
> So M_0 will get to step 2 and 3.
> When it does, it will output $y + 5$.
> Since M_0 outputs $y + 5$ for every input, there exists at least one input for which it outputs $y + 5$ (e.g., $y = 1$).
> $\Rightarrow \langle M_0 \rangle \in L_3$.

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

Then M loops on w.
Therefore M_0 loops on line 2.
So for every input, M_0 loops and does not output anything (in particular, it never outputs $y + 5$).
$\Rightarrow \langle M_0\rangle \notin L_3$.

Finally, we'll give talk about the language DEC. This is another more difficult example.

Do we think that DEC is recognizable, co-recognizable, both, or neither? Why?

> We're asking whether there is some decider N such that $L(M) = L(N)$. So we'd expect this to be at least as hard as EQ_{TM}.
>
> *In fact, it's harder... the fact that we're using three rather than two quantifiers is a bit of a tip-off.*

OK, with that done, let's get through the reductions.

This time we want our reductions to output TMs whose languages will vary between decidable and undecidable depending on whether M loops on w. So we'll want to start with some undecidable language, and build our M_0 around that.

Whatever language we use has to be undecidable but recognizable. Do you see why?

So let's use the language HALT.

This means that M_0 will take inputs that will be instances of HALT. To separate these instances from the $\langle M, w\rangle$ we're reducing from, we'll call these instances $\langle N, x\rangle$.

Here Are the reductions:

- **Proof that DEC is not recognizable**: $\overline{\text{HALT}} \leqslant_m \text{DEC}$:

  We want to set up M_0 so that it recognizes whether $\langle N, x\rangle \in \text{HALT}$ iff M loops on w. Now, if M loops on w it runs forever, while if it halts it stops after a finite number of steps. So we can use the number of steps it takes for M to halt to give an upper limit to how long to let N run on x — we do this by dovetailing. With this in mind, we get this reduction:

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

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

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

Then M loops on w.
So M_0 loops on either step 1 or step 2.
$\Rightarrow L(M_0) = \varnothing$.
$\Rightarrow \langle M_0 \rangle \in \text{DEC}$.

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

Then M halts on w.
Therefore M_0 passes line 2 and accepts iff it finishes step 1.
So M_0 halts and accepts iff N halts on x.
$\Rightarrow L(M_0) = \text{HALT}$.
$\Rightarrow \langle M_0 \rangle \notin \text{DEC}$.

- **Proof that** DEC **is not co-recognizable**: $\text{HALT} \leqslant_m \text{DEC}$:

The idea here is actually quite similar — we want M_0 to recognize HALT iff M halts on w. We can do this with something like the default reduction — use a step in which we run M on w as a filter that may prevent us from reaching later calculations.

Here's the reduction:

- Consider the TM P "On input $\langle M, w \rangle$:
  1. Construct the TM M_0 as follows:
     M_0 = "On input $\langle N, x \rangle$:
     1. For $k = 0$ to ∞:
     1. Run N on x for k steps.
     2. Run M on w for k steps.
     3. If either halts, *accept.* "
  2. Return $\langle M_0 \rangle$."

Proof that $\langle M_0 \rangle \in \text{DEC}$ **iff** $\langle M \rangle \in \text{HALT}$:

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

Then there is a k such that M halts on w in k steps.
So M_0 will accept by the k^{th} loop iteration, regardless of whether N halts on x.
$\Rightarrow L(M_0) = \Sigma^*$.
$\Rightarrow \langle M_0 \rangle \in \text{DEC}$.

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

Then M loops on w.
So M does not halt on w in k steps for any k.
Therefore M_0 accepts iff there is a k for which N accepts x within k steps.
So M_0 halts and accepts iff N halts on x.
$\Rightarrow L(M_0) = \text{HALT}$.
$\Rightarrow \langle M_0 \rangle \notin \text{DEC}$.

Here's an idea to remember: lots of our reductions have outputted M_0 that can loop. The reduction P itself has to halt, but there's nothing that requires its output to do the same.

And now have a bunch of reduction examples to work with. But you'll have noticed something interesting about the undecidable languages we've seen so far:

They all have to do with the languages of TMs.

This isn't too surprising: we got our first undecidability result by using TMs that use self-reference, and we've just built up from there. In fact, just about any statement about a TM's language is undecidable:

Rice's Theorem: (text, p.241)

Suppose that P is a language of TM descriptions such that:

- P is neither empty nor the set of all TM descriptions.
- If M_1 and M_2 are TMs such that $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$.

Then P is undecidable.

The language P has to be defined in terms of the *semantics* of the TM, not by its *structure*, though — this is why we can tell that a C program will or will not compile, even if we can't say what it will do.

But you can see that this theorem means that many of the interesting questions we can ask about TMs (or computer programs) will in general be undecidable.

Note: *Since part of the purpose of this course is to get you used to doing reductions, you are not allowed to use Rice's theorem in assignments or tests unless otherwise stated. You can use it, however, as a quick way of proving undecidability for when you're outside of CSCC63.*

This leads us to a bit of a problem, though — we've proven a number of languages are undecidable, but all of those languages have to do with figuring out what TMs or computer programs do.

There are times when we want to figure out what a program will do — when we're running automarkers on your assignments, for instance, or if we're working in computer security.

But one of the ideas we want you to get from this course is that undecidable programs can show up in many places, even if there is no obvious TM in the problem — if we look around (say, on YouTube), we can find TMs implemented in many different media: e.g., Lego, wooden machines, Minecraft, and so on.

If we can implement a TM in these media, what does this tell us about the complexity of working in said medium?

So if we can embed a TM into a system, some questions about that system will be undecidable.

Some important undecidable systems include:

1. **Hilbert's 10^{th} Problem:** Suppose that we have a multivariate polynomial with integer coefficients, and we want to tell if it has any root whose variables are integers.

   We can solve some instances of this problem:

   - We can solve it for linear polynomials.
   - We can solve it for quadratic polynomials (although this is quite a bit harder).
   - We can solve it for polynomials in one variable.

   But it was found in 1970 by Matiyasevich that the general problem is undecidable. Specifically, we refer to any multivarite polynomial P as **Diophantine**, and we refer to any set $S \in \mathbb{Z}^n$ as a **Diophantine set** iff there is a Diphantine polynomial P in $n + m$ variables such that

   $$S = \left\{(x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n \;\middle|\; \exists (y_1, y_2, \ldots, y_m) \in \mathbb{Z}^m, P(x_1, \ldots, x_n, y_1, \ldots, y_m) = 0\right\}.$$

   It turns out that the Diophantine sets are precisely the sets of $\mathbb{Z}^n$ that are recogniable using TMs.

2. **Recursive Function Theory.**

   This is one of the alternative formulations of computability that we talked about earlier in the course – it's the one built by Alonzo Church. We can show that there are a smll number of basic functions on the natural numbers that we can use as a basis for more complex functions in such a way that, mong other things, we can encode any TM as a partial function from $\mathbb{N}$ to $\mathbb{N}$ using this basis.

3. The **Post Correspondence Problem**: We'll go over this problem in detail next week, but you've seen it in tutorials.

Other undecidable problems include:

- General queries about cellular automata such as Conway's Game of Life,
- Tiling problems on the plane (if you play board games, think generalized Carcassonne),
- If you've taken group theory in math, the word problem on free groups is undecidable,
- And many more…

We won't go into detail about all of these, but we'll show the Post Correspondence problem next week, since it'll give you the idea of how TM embeddings work, and since it's a nice starting point for other undecidability proofs.

The Take-Away from this Lesson:

- We've introduced a few more languages and seen more mapping reductions.
- We've talked about (but not proven) Rice's theorem.
- We've talked about a few places without TMs where undecidable problems pop up.

Glossary:

Rice's Theorem

***Languages*:**

DEC, EQ_{TM}, L_3

A Thought:

Since we're moving from TM languages to non-TM languages, let's take a look back at what we've already seen.

We've used certificates as a way of talking about recognizable languages, but lots of the languages we've seen haven't been either recognizable nor co-recognizable. Can we generalize the idea of a certificate to cover these languages?

As a matter of fact, yes (sometimes)! For example, we've said that $\langle M, w, k \rangle \in \text{ACC}$ iff M accepts w in k steps, and so the language of ACC is decidable. But we've also seen that:

$$\text{E}_{\text{TM}} = \left\{ \langle M \rangle \,\middle|\, \forall w, k, \langle M, w, k \rangle \notin \text{ACC} \right\}$$

(co-recognizable)

$$\text{ALL}_{\text{TM}} = \left\{ \langle M \rangle \,\middle|\, \forall w \in \Sigma^*, \exists k \in \mathbb{N}, \langle M, w, k \rangle \in \text{ACC} \right\}$$

(neither – but we'll see soon that this is in a class we call Π_2)

$$\begin{aligned} L_2 &= \left\{ \langle M \rangle \,\middle|\, \forall w \neq 1, k, \exists k', \langle M, 1, k' \rangle \in \text{ACC} \wedge \langle M, w, k \rangle \notin \text{ACC} \right\} \\ &= \left\{ \langle M \rangle \,\middle|\, \exists k', \forall w \neq 1, k, \langle M, 1, k' \rangle \in \text{ACC} \wedge \langle M, w, k \rangle \notin \text{ACC} \right\} \end{aligned}$$

(neither – but we'll see soon that this is in a class we call Δ_2)

Do you see a pattern?

Basically, each extra quantifier adds an extra layer of complexity, and the type and order of the quantifiers matters. If you're interested, look up something called the **Arithmetic Hierarchy**. A language is in the Arithmetic Hierarchy iff it can be expressed as a decidable predicate with a finite number of certificates, each with its own quantifier.

We'll investigate this idea a little more in next week's lecture and a lot more in a handoout soon. Note that not all languages are part of the Arithmetic Hierarchy at all (some of the Kolmogorov complexity examples from Sipser qualify here).