Extra Notes: Oracles and the Arithmetic Hierarchy

The mapping reductions that we've been using to prove certain languages are not decidable can actually be used to give a lot more information than that. There's a lot of structure to the undecidable languages, and we'll look at some of that structure here.

1 What is an Oracle?

Here are a couple of problems to think about:

- Later on we'll run into problems that we think cannot be solved efficiently, but which we have not yet proven to be intractable. What would be the consequences if these problems were solvable?
- We've proven in class that there's no TM that'll solve HALT for us. According to the Church-Turing thesis this means that there's no way to solve it at all using any reasonable computer.

  But what about unreasonable computers?

  As an eample, suppose that tomorrow someone comes up with a very surprising announcement: a bunch of new physical laws have been discovered which will allow us to build computers that solve HALT!

  How would this change the computability landscape?

While we strongly believe the Church-Turing thesis to be true, and while we strongly believe (without proof) that some of the problems we'll run into later are intractable, it can be good to at least address these types of questions. That's where the idea of an **oracle** comes in.

Imagine this: *I've gone off to a garage sale somewhere, and among the items for sale I see a crystal ball. It's labeled as a magic crystal ball. I go to the counter and ask what kind of magical crystal ball it is. The vendor tells me:*

"This ball will solve the halting problem for you. Just ask if whether an $\langle M, w\rangle \in$HALT and it'll always answer your question correctly. A warning, tohugh – since magic is arbitrary, it won't help you with A_{TM}, ALL_{TM}, or the stock market."

"But wait, I thought that there was a proof that no computer program can solve HALT?"

"That's true, no computer can solve it. But the proof doesn't cover magical crystal balls you buy at garage sales."

Natually, I see no problem with this logic, and so I immediately buy the crystal ball. Once I get home, I try it out and find that it works (at least, as far as I can tell...).

But I can never leave well enough alone, so I plug it into my computer just to see what happens. I now have a computer that'll solve HALT (but not ALL_{TM} or stock market questions).

In this rather silly example, the crystal ball would be an oracle for HALT. It's really just an unknown device that'll do the work of solving the problem for us.

1.1 Oracles: A Slightly More Formal Definition

The idea here is that an oracle is a widget we can add to our TMs (alternatively, our computers) that is not itself a TM, but which can be used to answer questions about a specific language.

Formally, if we want to use an oracle for a language $L \subseteq \Sigma^*$ with a TM we build an **Oracle Turing Machine** (an OTM). An OTM has a definition just like a regular TM, except we add an extra tape (the oracle tape), and two distinguished states $q_?$ (the query state) and $q_!$ (the answer state). The oracle tape is initially filled with blank symbols, and can be overwritten as part of the usual TM operation.

If my TM enters the query state, then:

- If the string on the oracle tape is a member of L, then the leftmost cell is overwritten to a 1. If it is not in L, then it is overwritten to a 0.
- Every non-blank symbol on the oracle tape except for the leftmost symbol is erased and replaced with the blank symbol.
- The head of the oracle tape is moved to the leftmost cell.
- The OTM enters the $q_!$ state.

Note that all of these actions occur in a single TM step. We'll say that the set of strings accepted by an OTM is its langauge, that it recognizes this language using its oracle, and in addition that it decides its language using its oracle if it halts on all of its inputs.

> *It's really the same as the regular TMs we use, except for the interface that lets us query the oracle.*

Furthermore, we can see that each of these OTMs can be represented as a string, much like we can represent any TM as a string. This means that we'll write, e.g.,

$$\langle M^L \rangle = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R, q_?, q_!)$$

when specifying the OTM.

Note that just as we can use a universal TM $M_{\mathbf{U}}$ to emulate any other TM M given its string description $\langle M \rangle$, we can write a universal OTM $M^L_{\mathbf{U}}$ using an oracle for a language L to emulate any other OTM M^L using an L oracle, so long as we have its string description $\langle M^L \rangle$. The construction we'll use to manage this is essentially the same as the one we used in class.

1.2 Using Oracles in OTMs

Practically, though, we usually write pseudocode that we argue that we can compile onto a TM, and we do the same for OTMs: we'll write code that is allowed to make oracle calls. Let's go over a few examples of this.

If I take my oracle from the garage sale and plug it into my computer, I can obviously solve HALT. But I can also write programs that'll solve other languages, like $\mathrm{A_{TM}}$ or $\mathrm{E_{TM}}$.

Oracle program to solve $\mathrm{A_{TM}}$:

```
Let $M^{\mathrm{HALT}}_{\mathrm{A_{TM}}}$ = "On input $\langle M, w\rangle$:
  1. Query the oracle on $\langle M, w\rangle$.
  2. If $M$ halts on $w$ (from the oracle response):
  3.     Run $M$ on $w$.
  4.     If it accepts, *accept*.
  5. *Reject*."
```

In fact, we can solve some surprising languages this way: remember the language L_2 from our mapping reduction examples:

$$L_2 = \left\{\langle M\rangle \,\middle|\, L(M) = \{1\}\right\}.$$

We showed that this language is neither decidable, recognizable, nor co-recognizable. But we could decide it if we had an oracle for HALT:

Oracle program to solve L_2:

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

```
Let $M^{\mathrm{HALT}}_{L}$ = "On input $\langle M\rangle$:
  1. Let $M_1$ ="On input $x$:
                1. Run $M$ on 1.
                2. If it accepts, *accept*, otherwise *loop*."
  2. Let $M_2$ ="On input $x$:
                1. For $k = 0$ to $\infty$:
                2.     For $i = 0$ to $k$:
                3.         If $w_i \neq 1$:
                4.             Run $M$ on $w_i$ for $k$ steps.
                5.             If it accepts, *accept*."
  3. Query the oracle on both $\langle M_1, \varepsilon\rangle$ and $\langle M_2, \varepsilon\rangle$:
  4. If $M_1$ halts on $\varepsilon$ and $M_2$ does not halt on $\varepsilon$:
  5.     *Accept*.
  6. *Reject*."
```

So I can solve a lot more problems with my computer with the oracle than without. But not every problem: I'd find that $\mathrm{ALL_{TM}}$ is co-recognizable but not recognizable for my new computer.

In fact, suppose I tried writing the following program:

```
Let $M^{\text{HALT}}$ = "On input $\langle M, w\rangle$:
  1. If $M$ halts on $w$ (use the oracle here):
  2.     *Accept.*
  3. Else:
  4.     *Accept.*"
```

Now, I feed $\langle M^{\text{HALT}}, w\rangle$ into my oracle for some input w, and start getting weird results — it might claim, for example, that the program loops, when it should clearly halts! After looking at it for a while, though, I realize the problem:

The oracle assumes the M^{HALT} it is seeing is a regular TM: one without an an oracle attached. So when it sees the oracle call, it's looking at an input that doesn't match its expected type. Just like in a regular computer program, if you feed in input that is outside of the expected preconditions, you can get unexpected results – the oracle behaviour is undefined. In my example here I had it reject the bad inputs.

So the oracle can't directly answer questions about other computers that are using the same oracle. The question is, can we write a clever program like we did for L_2 that will, for example, tell us whether another oracle TM halts?

You'll find my oracle computer still can't answer its own halting problem, even though it can solve the halting problem for a computer without the oracle. In fact, the same diagonalization argument we used for HALT "lifts" to this domain as well! (We sat that this sort of diagonalization argument *relativizes.*)

> You'll have noticed that above we've written a few different programs with oracle calls, and we've denoted each of them with the little $^{\text{HALT}}$ superscript. That's there to indicate that we're looking at a program that uses a bit more power than we'd normally have. It tells me that the "If M halts/loops on w" conditions aren't mistakes, but are allowed because we're assuming we have a widget that'll answer the question for us.
>
> - TMs/computer programs without superscripts are regular programs that we can build on real computers: e.g., $M, N, \ldots$
> - TMs/computer programs with superscripts are OTM/oracle programs with access to an oracle for whichever language is in the superscript: e.g., M^{HALT} can condition on HALT, while $N^{\text{ALL}_{\text{TM}}}$ can condition on ALL_{TM}.
>
> We can do the same thing with languages: the superscript indicate that the programs in the language use an oracle. E.g.:

$$\text{HALT}^{\text{HALT}} = \left\{ \langle M^{\text{HALT}}, w\rangle \;\middle|\; M^{\text{HALT}} \text{ halts on } w \right\}.$$

Let's use this terminology to help us prove that no OTM can solve its own halting problem:

Theorem 1.1 *Let $L \subseteq \Sigma^*$ be any language, and let HALT^L be the halting problem for the set of OTMs on L:*

$$\text{HALT}^L = \left\{ \langle M^L, w\rangle \;\middle|\; M^L \text{ halts on } w \right\}.$$

Then there is no OTM N^L that decides HALT^L.

Proof:

Suppose to the contrary that there is some N^L that decides HALT^L. Then consider the following OTM:

```
$D^L$ =“On input $\langle M^L \rangle$:
  1. Run $N^L$ on $\langle M^L, M^L \rangle$.
  2. If it accepts, loop forever.
  3. Otherwise, *accept*.
```

Now, consider what happens if we run D^L in the input $\langle D^L \rangle$:

- **If D^L loops on itself**, then N^L will not accept on line 1. But N^L is by assumption a decider, and so it must reject. Specifically, this means that line 1. must eventually terminate.

  This means that D^L reaches the conditional in line 2., and since N^L must reject, we must go to line 3.

  Once we reach line 3., D^L will accept and halt, a contradiction.
- **On the other hand, if D^L halts on itself**, then N^L will accept on line 1. We will then proceed to the conditional on line 2, and since N^L has accepted D^L will loop forever, a contradiction.

In either case we must have a contradiction, and so our assumption that N^L exists must be mistaken.

So HALT^L cannot be decided by an OTM with access to an oracle to L, as required.

As we have said, this is simply a “lift” (called a *relativization*) of the usual diagonalization argument to show that HALT is undecidable – you can see that this is very similar to the argument for $\mathrm{A_{TM}}$ that we used in class.

We can relativize the definitions of decidability, recognizability, and so on, as well: we say that an OTM M^L decides a language L' using an L oracle if M^L halts on all inputs and if $L(M^L) = L'$. In this case we say that L' is decidable relative t L. Similarly, if $L(M^L) = L'$ (note that we allow M^L to loop here), then we say that M^L recognizes L' using an L oracle, and that L' is recognizable relative to L.

In fact, another proof relativizes quite nicely: our argument that a language is decidable iff it is both recognizable and co-recognizable. The same argument can be used to show that, e.g., a language is decidable using a HALT *oracle iff it is both recognizable and co-recognizable using a* HALT *oracle. We'll leave the proof as an exercise for the reader and make use of the fact in the following sections.*

2 Oracles and Reductions

We have seen that OTMs are more powerful than regular TMs, since any regular TM calculation history can be emulated on an OTM: we just treat the TM as an OTM that happens to not use any oracle calls. But what just how strong are OTMs – to what extent can we bound, e.g., the power of a TM with a HALT oracle, in terms of the languages that we're already familiar with? Can we build an OTM with a HALT oracle, for example, to decide $\mathrm{ALL_{TM}}$?

The answer turns out to be no, *as we'll see shortly.*

But to prove this sort of result we'll need to have some way of dealing with the oracles in our OTMs. They give us extra power to work with, but that extra power also means that it's that much harder to build reductions.

One way to deal with this difficulty is to observe that we can, to some extent, *emulate* OTMs with regular TMs (or other OTMs). Specifically, given a language L and an OTM M^L, let's look at how we could build a TM N which would (among other things) simulate M^L on input w. Let $\hat{M}$ be the simulated image of M^L in N. We will, in general, refer to the true configuration of M^L on step i as C_i^L, and the simulated configuration in $\hat{M}$ on step i as $\hat{C}_i$. Our goal will in general be to set up N so that there are reasonable situations in which we can guarantee that each $C_i^L = \hat{C}_i$. In order to do so, let's first write the follwoing:

- The contents of the oracle tape of M on step i of input w will be referred to as $\psi_{M,w,i}$.

  Note that this definition applies to both the true OTM M^L and to the simulated OTM $\hat{M}$. The contents of the oracle tape of M^L on step i when run on w is $\psi_{M^L,w,i}$, and for $\hat{M}$ it is $\psi_{\hat{M},w,i}$. The same distiction will apply to the following variables as well.

- The oracle response for M on step i when run on w will be referred to as $\alpha_{M,w,i}$. Note that the oracle response is not defined unless there is ctually an oracle call in step i (i.e., if M is in the state $q_?$).

- When the oracle response is defined, it is a binary response. We can take these responses and conctenate them to form a string $A_{M,w,i}$. This string tells us the oracle responses that M sees in its firsst i steps when run on w. Note that in general $|A_{M,w,i}| < i$ – if we run M for 100 steps and make 10 oracle calls, then $|A_{M,w,i}| = 10$.

- In a similar vein, we can male a list of the contents of the oracle calls made by M in its first i steps. If we run M for i steps on input w and record the contents of the oracle tape during the steps in which we are in the state $q_?$, the result will be the list $\Phi_{M,w,i}$.

- The oracle (or our simulated oracle stand-in) will predict some of these oracle queries to be *yes*-instances and some to be *no*-instances. We'll denote the sublist of $\Phi_{M,w,i}$ made up of predicted *yes*-instances as $\Phi^+_{M,w,i}$ and the predicted *no*-instances as $\Phi^-_{M,w,i}$.

N will proceed as follows:

- We will initialize $\hat{M}$ to have the initial configuration $\hat{C}_0 = q_0 w$, where q_0 is the initial state of M^L.

  Since C_0^L is the true initial configuration of M^L on w, then we can see that $\hat{C}_0 = C_0^L$.

- If, in the time step i, $\hat{M}$ is in a configuration $\hat{C}_i$ with a state $q_i \neq q_?$, then N will update $\hat{M}$ using the usual TM rules to get a new simulated configuration $\hat{C}_{i+1}$.

  If C_i^L and C_{i+1}^L are the true configurations of M^L on w, and if $\hat{C}_i = C_i^L$, then clearly $\hat{C}_{i+1} = C_{i+1}^L$.

- On the other hand, if $\hat{C}_i$ has $\hat{M}$ in the state $q_?$, then N will (after possibly copying the configuration $\hat{C}_i$ into memory) simulate the state $\hat{C}_{i+1}$ as follows:
    - The oracle tape $\psi_{\hat{M},w,i}$ of $\hat{M}$ will be erased (possibly after copying to elsewhere).
    - The initial cell of the oracle tape for $\hat{M}$ will be set to some $\alpha_{\hat{M},w,i} \in \{0,1\}$.
    - The oracle tape head will be moved to the leftmost cell.
    - The state of $\hat{M}$ will be set to $q_!$.

  If C_i^L and C_{i+1}^L are the true configurations of M^L on w, and if $\hat{C}_i = C_i^L$, then it may be that $\hat{C}_{i+1}$ is <u>not</u> the same as C_{i+1}^L. However, if we can fortuitously ensure that $\alpha_{\hat{M},w,i}$ happens to match the oracle's true response, then it does follow that $\hat{C}_{i+1} = C_{i+1}^L$

We can easily see using induction on i that

Proposition 2.1 *The computation history of the simulated $\hat{M}$ matches the true computation history of M^L on w iff $\alpha_{\hat{M},w,i} = \alpha_{M^L,w,i}$ for all i such that both $\hat{M}$ and M^L are in the state $q_?$, and that this occurs iff $\psi_{\hat{M},w,i} = \psi_{M^L,w,i}$ for all $i \in \mathbb{N}$.*

□

2.1 Comparing Different Oracles

Let's look at an example of how to use this idea:

Theorem 2.2 *Suppose that $L \leqslant_m L'$, and let M^L decide (resp. recognize/co-recognize) some language X. Then there is some OTM $N^{L'}$ that also decides (resp. recognizes/co-recognizes) X.*

Proof:

Suppose that $L \leqslant_m L'$. Then, by definition, there is some computable function $f : \Sigma^* \mapsto \Sigma^*$ such that for all $x \in \Sigma^*$, $f(x) \in L'$ iff $x \in L$. Let $N^{L'}$ be defined as follows:

```
Let $N^{L'}$ ="On input $w$:
  1. Let $\hat{M}$ be a simulated version of $M^L$, set to an initial configuration $q_0 w$.
  2. While $\hat{M}$ is not in a halting state:
  3.   If $\hat{M}$ is *not* in the state $q_?$:
  4.     Update $\hat{M}$ normally.
  5.   Else:
  6.     Let $x = O_{\hat{M},w,i}$.
       *Recall that this just means that we copy the oracle tape to $x$.*
  7.     Erase the oracle tape of $\hat{M}$,
         set its oracle tape head to the leftmost cell, and
         set its configuration to $q_!$.
  8.     Query the oracle for $L'$ on $f(x)$.
  9.     Set the oracle tape for $\hat{M}$ to the $L'$ oracle response.
  10. If $\hat{M}$ is in an accept state, *accept*. Otherwise *reject*.
```

Now, suppose that, in step i of our computation, the string x on the oracle tape of $\hat{M}$ matches the corresponding string in M^L and that both TMs are in state $q_?$. We can see that $\alpha_{\hat{M},w,i} = 1$ iff $f(x) \in L'$, which occurs iff $x \in L$, by the definition of a mapping reduction. Hence, $\alpha_{\hat{M},w,i} = \alpha_{M^L,w,i}$.

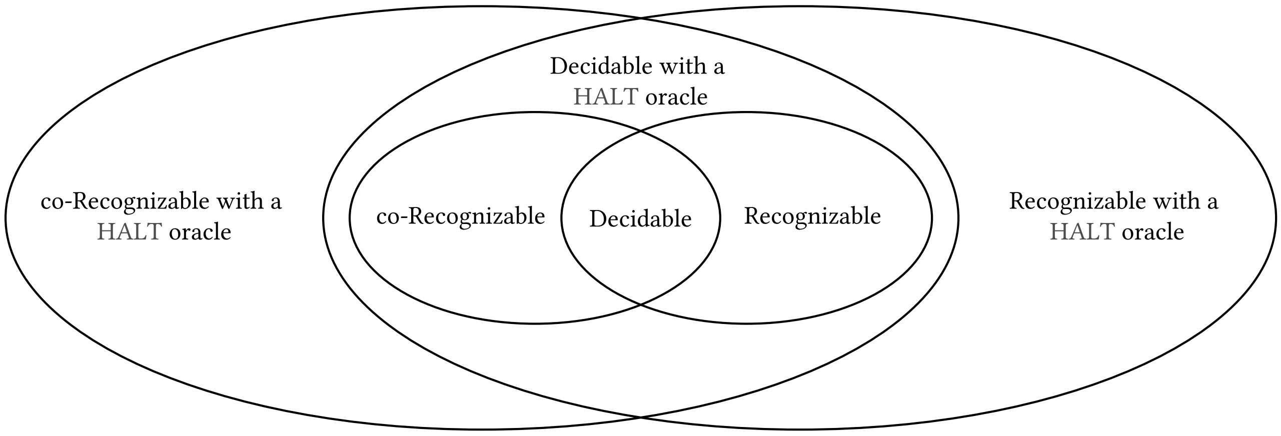
The result now follows from 2.1.

■

So if $L \leqslant_m L'$, then the OTMs using an oracle for L' have at least as much power as those that use oracles for L. We can resonably what happens if we build OTMs using increasingly powerful oracles – e.g., OTMs that use oracles for HALT, for $\text{HALT}^{\text{HALT}}$, for $\text{HALT}^{\text{HALT}^{\text{HALT}}}$, and so on. This is indeed the case, as we'll see in the next section.

3 The Arithmentic Hierarchy

We already know that we have languages that are decidable, recognizable, and co-recognizable (with no oracle). What's more, we've seen that if we have an oracle for HALT, we can decide any language from any of these three classes, and more (see our HALT oracle decider for L_2 in 1.2). But we also know from 1.1 that there is no OTM using a HALT oracle that can decide $\text{HALT}^{\text{HALT}}$. Furthermore, we can see that $\text{HALT}^{\text{HALT}}$ is *recognizable* using a HALT oracle. In fact, the class of languages that are recognizable/co-recognizable using a HALT oracle are a strict superclass of the languages that are decidable using a HALT oracle, and that this class is a strict supoerclass of the classes of recognizable or co-recognizable languages. So the classes we know about look something like this:


Decidable with a HALT oracle
co-Recognizable with a HALT oracle
co-Recognizable
Decidable
Recognizable
Recognizable with a HALT oracle

And there's no real reason we can't keep adding to these classes: we can define a class of languages that are decidable with an oracle for $\text{HALT}^{\text{HALT}}$, the languages that are recognizable or co-recognizable with an oracle for $\text{HALT}^{\text{HALT}}$, and so on. Theorem 2.2 ensures that every time we add a more powerful oracle we end up with a strict superclass of languages.

Let's make this exact – we'll define a strictly increasing set of language classes Δ_i, Σ_i, and $\Pi_i, i \in \mathbb{N}$. Our definintions will be inductive, starting with the decidable languages:

$$\Delta_0 = \Sigma_0 = \Pi_0 = \{L \mid L \text{is decidable}\}.$$

For $i > 0$ and any language L, we'll say that:

- $L \in \Delta_i$ iff there is a decider for L that uses an oracle for some language in Σ_{i-1}.
- $L \in \Sigma_i$ iff there is a recognizer for L that uses an oracle for some language in Σ_{i-1}.
- $L \in \Pi_i$ iff there is a co-recognizer for L that uses an oracle for some language in Σ_{i-1}.

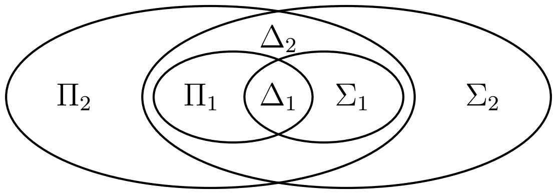
Using these definitions we see that Δ_1 is the class of languages using an oracle for any decidable language L. But, of course, if L is decidable we really don't need the oracle for it, we can just compute it. So $\Delta_1 = \Delta_0$ (we haven't done anything interesting just yet). Similarly, Σ_1 is the class of languages that are recognizable, and Π_1 is the class of languages that are co-recognizable.

Now, suppose that L' is decidable using an oracle for some language $L \in \Sigma_1$. So L is recognizable. But you've seen in your week 4 tutorial that, if L is recognizable then $L \leqslant_m \mathrm{A_{TM}} \leqslant_m$ HALT. So, using 2.2 we can see that L' is also decidable using an oracle for HALT. Conversely, if L'' is decidable using an oracle for HALT, then it is in Δ_2, since HALT is recognizable, and therefore in Σ_1. Therefore, Δ_2 is precisely equal to the class of langauges decidable using a HALT oracle.

We can continue with the same line of thought to find that:

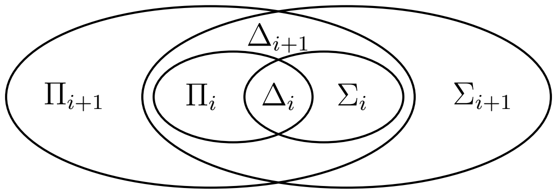
- Σ_2 is the class of languages that is recognizable with an oracle for HALT. The language $\mathrm{HALT}^{\mathrm{HALT}}$ is in this class.
- Π_2 is the class of languages that is co-recognizable with an oracle for HALT.
- Δ_3 is the class of languages that is decidable with an oracle for $\mathrm{HALT}^{\mathrm{HALT}}$.
- Σ_3 is the class of languages that is recognizable with an oracle for $\mathrm{HALT}^{\mathrm{HALT}}$.
- Π_3 is the class of languages that is co-recognizable with an oracle for $\mathrm{HALT}^{\mathrm{HALT}}$.
- Δ_4 is the class of languages that is decidable with an oracle for $\mathrm{HALT}^{\mathrm{HALT}^{\mathrm{HALT}}}$.
- ...

With this in mind, our previous diagram can be written as:



Note that if a class A is depicted in this diagram as being contained in another class B, then A is in fact a proper subclass of B (E.g., Π_1 is a proper subclass of Σ_2, as depicted in the picture).

And this property extends to all i – if we look at any classes A and B in the following picture:



Then if A is contined in B according to the picture, then it is proper subclass of B.

To illustrate this, we note first of all that $\Delta_1 \subseteq \Sigma_1 \subseteq \Delta_2$: if a language is decidable using no oracles, then it is decidable using a HALT oracle as well. A TM using no oracle can, after all, be thought of as an OTM that just happens to never use an oracle.

Note that if L is decidable using an oracle for a language L, then it is also recognizable using the same oracle. So $\Delta_i \subseteq \Sigma_i$ for all $i \in \mathbb{N}$. It follows that if $\Delta_{i-1} \subseteq \Sigma_{i-1} \subseteq \Delta_i$, then $i \in \mathbb{N}$. It follows that if $\Delta_{i-1} \subseteq \Sigma_{i-1} \subseteq \Delta_i \subseteq \Sigma_i$. So if a language L' is decidable using an oracle for some language $L \in \Sigma_{i-1}$, it must also be decidable using the language $L \in \Sigma_i$. Hence, $L' \in \Delta_{i+1}$. So $\Delta_{i-1} \subseteq \Sigma_{i-1} \subseteq \Delta_i$ implies that $\Delta_i \subseteq \Sigma_i \subseteq \Delta_{i+1}$.

We can then use induction to show that $\Delta_i \subseteq \Sigma_i \subseteq \Delta_{i+1}$ for all $i \in \mathbb{N}$. A similar argument can also be used to show that $\Delta_i \subseteq \Pi_i \subseteq \Delta_{i+1}$ for all $i \in \mathbb{N}$.

Let's show that these containments are strict. To do so, we'll make the follwing inductive definition:

- $\mathrm{HALT}^0 = \{\varepsilon\}$

  In fact, any decidable language other than $\varnothing$ or Σ^* would do.
- For $i > 0$, $\mathrm{HALT}^i = \mathrm{HALT}^{\mathrm{HALT}^{i-1}}$.

Clearly, $\mathrm{HALT}^1 = \mathrm{HALT}$. Thus, we have seen that $\mathrm{HALT}^1 \in \Sigma_1$ and for all $L \in \Sigma_1, L \leqslant_m \mathrm{HALT}^1$.

This means that:

Theorem 3.1 *The langauge HALT^i satisfies the following properties*[1]*:*

- $\mathrm{HALT}^i \in \Sigma_i$, *and*
- *for any $L \in \Sigma_i$, $L \leqslant_m \mathrm{HALT}^i$.*

Proof:

We prove this using induction.

Base Case: $n = 1$. We have already proven that $\mathrm{HALT}^1 = \mathrm{HALT} \in \Sigma_1$ and that, for any $L \in \Sigma_1$, $L \leqslant_m \mathrm{HALT}^1$. So the base case holds.

[1]We'll introduce the notion of class *hardness* and *completeness* in the next few weeks. Wht this theorem actually says is that HALT^i is Σ_i-complete.

Induction Step: Suppose that $i \geqslant 1$ and that

- $\mathrm{HALT}^i \in \Sigma_i$, and
- for any $L \in \Sigma_i$, $L \leqslant_m \mathrm{HALT}^i$.

Since $\mathrm{HALT}^i \in \Sigma_i$, $\mathrm{HALT}^{i+1} = \mathrm{HALT}^{\mathrm{HALT}^i}$ is recognizable with a HALT^i oracle: given an OTM/input pair $\langle M^{\mathrm{HALT}^i}, w\rangle$, we simply run M^{HALT^i} on w until it halts, and of it does, we accept. So $\mathrm{HALT}^{i+1} \in \Sigma_{i+1}$.

Now, suppose that $L \in \Sigma_{i+1}$. Then there is some recognizer R^{HALT^i} for L. If we set

```
$M^{\mathrm{HALT}^i}$ =“On input $w$:
    1. Run $R^{\mathrm{HALT}^i}$ on $w$.
    2. If if accepts, accept, otherwise loop.”
```

Now, we can see that M^{HALT^i} accepts a string w iff R^{HALT^i} also accepts it, and so $L(M^{\mathrm{HALT}^i}) = L(R^{\mathrm{HALT}^i})$. Moreover, if M^{HALT^i} does not accept w then it will loop, and so

$$w \in L \Leftrightarrow R^{\mathrm{HALT}^i} \text{ accepts } w \Leftrightarrow R^{\mathrm{HALT}^i} \text{ accepts } w \Leftrightarrow \langle M^{\mathrm{HALT}^i}, w\rangle \in \mathrm{HALT}^{\mathrm{HALT}^i} = \mathrm{HALT}^{i+1}.$$

Thus, given a string $w \in \Sigma^*$, the computable function $w \mapsto \langle M^{\mathrm{HALT}^i}, w\rangle$ gives a mapping reduction $L \leqslant_m \mathrm{HALT}^{i+1}$, as required.

So the induction step holds.

Since both the base case and the induction step hold, HALT^i satisfies the required properties for all $i \geqslant 1$.

■

Note that since we do not allow HALT^0 *to be either* $\varnothing$ *or* Σ^**, the theorem holds even for* $i = 0$.

What this means is that if L' is decidable (resp. recognizable, co-recognizable) using an oracle for $L \in \Sigma_i$, then $L \leqslant_m \mathrm{HALT}^i$, and so, by 2.2, L' is decidable (resp. recognizable, co-recognizable) using an oracle for HALT^i. So we can characterize Δ_{i+1} (resp. Σ_{i+1}, Π_{i+1}) as the class of languages decidable (resp. recognizable, co-recognizable) using an oracle for HALT^i. But this implies:

1. Since $\mathrm{HALT}^i \in \Sigma_i$, and since no OTM can solve its own halting problem (from 1.1), we see that $\Delta_i \subsetneq \Sigma_i$: the containment must be strict.

2. If L is both recognizable and co-recognizable using an oracle for HALT^{i-1}, then it must be decidable using the same oracle – as we have said, this is just a restatement of the dovetailing argument for regular TMs.

   So, since $\mathrm{HALT}^i \in \Sigma_i$ but $\mathrm{HALT}^i \notin \Delta_i$, it follows that $\mathrm{HALT}^i \notin \Pi_i$, as well. So $\Sigma_i \neq \Pi_i$.

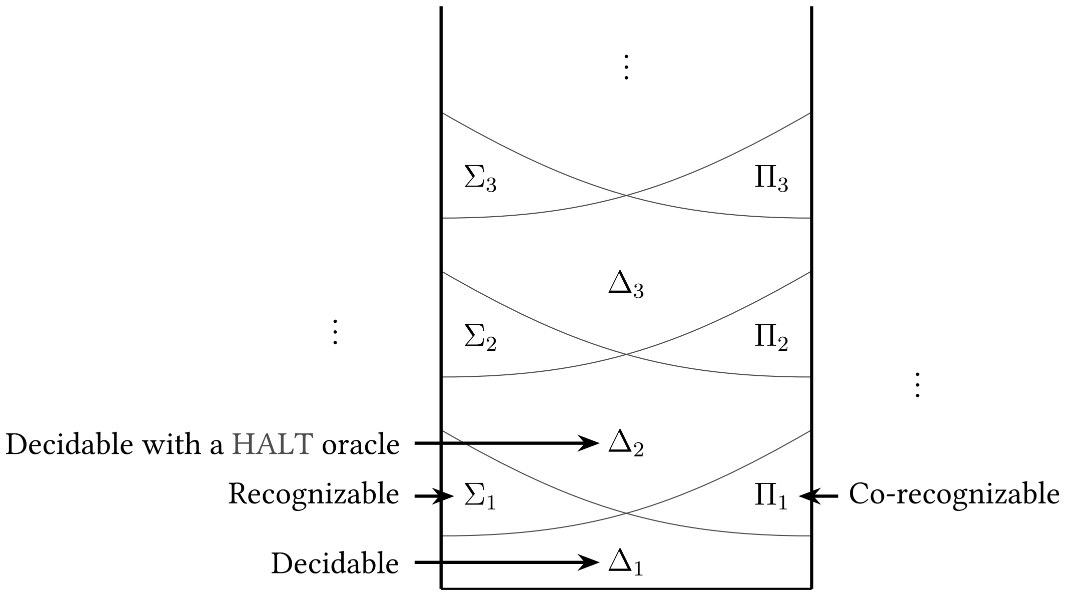
3. Since $\Sigma_i \subsetneq \Sigma_i \cup \Pi_i \subseteq \Delta_{i+1}$, it follows that $\Sigma_i \subsetneq \Delta_{i+1}$. Similarly, $\Pi_i \subsetneq \Delta_{i+1}$.

Putting all of these facts together we find:

Corollary 3.1.1 *For any $i \in \mathbb{N}$, $\Delta_i \subsetneq \Sigma_i \subsetneq \Delta_{i+1}$ and $\Delta_i \subsetneq \Pi_i \subsetneq \Delta_{i+1}$.*

We leave it to the reader as an exercise to show that each Δ_i is closed under complementation, and that each Δ_i, Σ_i, and Π_i is closed under the union, intersection, concatenation, and Kleene star operations.

What we end up with is an infinite tower of distinct language classes:



This tower of language classes is called the **Arithmentic Hierarchy**.

We can locate the different languages we've covered in lecture in these language classes as follows:

$\mathbf{\Delta_1}$ contains the languages ACC and H.

$\mathbf{\Sigma_1}$ contains the languages $\mathrm{A_{TM}}$, HALT, L_1, and L_3.

$\mathbf{\Pi_1}$ contains the language $\mathrm{E_{TM}}$.

$\mathbf{\Delta_2}$ contains the language L_2.

$\mathbf{\Sigma_2}$ contains the language FIN.

$\mathbf{\Pi_1}$ contains the languages $\mathrm{ALL_{TM}}$ and $\mathrm{EQ_{TM}}$.

$\mathbf{\Sigma_3}$ contains the language DEC.

Note that if a language is in any one of these classes then it is also in the any higher class – H is Δ_1, and so also Σ_1, Δ_2, Π_3, and so on. Similarly, HALT is in Σ_1, Δ_2, nd so on (but not in Δ_1 or Π_1).

We won't prove our claims about most of these languages here (you can try them as exercises), but we'll look at $\mathrm{ALL_{TM}}$ as an example of how to work with these higher classes.

4 Example: ALL_{TM} and Π_2

What we'll show here is that:

1) $\text{ALL}_{\text{TM}} \in \Pi_2$ and

2) For any language $L \in \Pi_2$, $L \leqslant_m \text{ALL}_{\text{TM}}$.

   This will show that Π_2 *is as low as we can place* ALL_{TM}. *When we talk about completeness this result will also be useful.*

To start off, we can see that $\text{ALL}_{\text{TM}} \in \Pi_2$ – in fact, we can write a recognizer for $\overline{\text{ALL}_{\text{TM}}}$ using a HALT oracle:

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

Let R^{HALT} =“On input $\langle M \rangle$:

```
1. For $i = 0$ to $\infty$:
2.     Query the HALT oracle on $\langle M, w_i \rangle$.
3.     If it loops, accept.
4.     Else:
5.         Run $M$ on $w_i$.
6.         If it rejects, accept.”
```

We can see that every iteration of the loop from lines 2. to 6. will eventually finish, and so:

- If $\langle M \rangle \in \overline{\text{ALL}_{\text{TM}}}$, then there is some string w on which M either rejects or loops. Let w_{i_0} be the first such string in our effective enumeration. Then, when run on $\langle M \rangle$, R^{HALT} will eventually reach the loop iteration $i = i_0$. In this iteration, if M loops on w_{i_0} then R^{HALT} will accept on line 3, while if M rejects w_{i_0} then R^{HALT} will accept on line 6. Either way, R^{HALT} will accept.

- If $\langle M \rangle \in \text{ALL}_{\text{TM}}$, then M will accept every string w. So for every $i \in \mathbb{N}$, the oracle call on line 2. of R^{HALT} will indicate that M halts on w_i, and so R^{HALT} will never accept on line 3. On the other hand, M will always finish running on w_i in line 5. and it will never reject. So R^{HALT} will not accept in line 6. Since this is true for all i, and since these are the only lines in which R^{HALT} can accept, we can see that R^{HALT} will never accept $\langle M \rangle$.

So $L(R^{\text{HALT}} = \overline{\text{ALL}_{\text{TM}}})$, and therefore $\overline{\text{ALL}_{\text{TM}}} \in \Sigma_2$. It follows that $\text{ALL}_{\text{TM}} \in \Pi_2$.

In order to finish our example, then, we need to show that $L \leqslant_m \text{ALL}_{\text{TM}}$ for all $L \in \Pi_2$, or equivalently, that $L \leqslant_m \overline{\text{ALL}_{\text{TM}}}$ for all $L \in \Sigma_2$. But in order to show this, we only really need to consider the language $\text{HALT}^{\text{HALT}}$, since we know from 3.1 that for any $\overline{L} \in \Sigma_2$, $\overline{L} \leqslant_m \text{HALT}^{\text{HALT}}$.

Theorem 4.1 *Let* $L = \mathrm{HALT}^{\mathrm{HALT}}$*. Then* $\overline{L} \leqslant_m \mathrm{ALL}_{\mathrm{TM}}$.

Before we move on to our proof, let's look at what this theorem says. The instances of $\overline{\mathrm{HALT}}^{\mathrm{HALT}}$ are OTM/input pairs $\langle M^{\mathrm{HALT}}, w\rangle$, and the yes-instances are the pairs for which M^{HALT} loops on w. Note that M^{HALT} is an OTM that is allowed to make calls to an oracle for HALT. But when we say that $\overline{\mathrm{HALT}}^{\mathrm{HALT}} \leqslant_M \mathrm{ALL}_{\mathrm{TM}}$ we're saying that there's some computable function f that transforms pairs $\langle M^{\mathrm{HALT}}, w\rangle$ into regular TM descriptions $\langle N\rangle$, where N accepts every input iff M^{HALT} loops on w. The important thing to remember is that neither the computable function f nor the output TM N may use an oracle, so we're going to need to find some way of handling the oracle calls in M^{HALT}. Let's have a look at how to do this:

Proof:

Let an instance $\langle M^{\mathrm{HALT}}, w\rangle$ of $\overline{\mathrm{HALT}}^{\mathrm{HALT}}$ be given.

The following reduction will follow the same basic strategy as the $\overline{\mathrm{HALT}} \leqslant_m \mathrm{ALL}_{\mathrm{TM}}$ *reduction: we'll try to demonstrate that* M^{HALT} *halts on* w*, and we'll accept if we fail to do so. The input to our TM* N *will be the parameters that we'll use in our tests.*

```
Let $N$ ="On input $\langle k \in \mathbb{N}, A \in \{0,1\}^*\rangle$:
  1. Let $\Phi^+ = []$ and $\Phi^- = []$.
     *$\Phi^+$ and $\Phi^-$ will be treated as lists containing the strings used in the oracle calls.*
  2. Let *oracle-count* $= 0$.
     *This is a counter to keep track of how many oracle calls have been made.*
  3. Let $M'$ be a simulated $M^{\mathrm{HALT}}$, set to the initial configuration $q_0w$.
  4. For $i = 1$ to $k$:
  5.    If $M'$ is not in the state $q_?$, update it normally.
         *If $M'$ is in a halting state, pass.*
  6.    Else:
  7.       Let $b = A[$*oracle-count*$]$.
           *If oracle-count$\geqslant |A|$, halt and accept.*
  8.       If $b$:
  9.          Append the oracle tape of $M'$ to $\Phi^+$.
  10.      Else:
  11.         Append the oracle tape of $M'$ to $\Phi^-$.
  12.      Erase the oracle tape of $M'$.
           and set the first cell to $b$.
  13.      *oracle-count*$++$.
  14.      Move the oracle tape head of $M'$ to the leftmost cell and set the state
  to $q_!$.
  15. If *oracle-count*$! = |A|$ or if $M'$ is not in an accepting state, *accept.*
  16. For each $\langle M_i, w_i\rangle$ in $\Phi^+$:
  17.    Run $M_i$ on $w_i$ for $k$ steps.
  18.    If it does not halt, halt and *accept.*
  19. For $j = 0$ to $\infty$:
```

```
20.     For $\langle M_i, w_i \rangle$ in $\Phi^-$:
21.         Run $M_i$ on $w_i$ for $j$ steps.
22.         If it halts, halt and *accept*.
23. Loop forever."
```

Return $\langle N \rangle$.

Now, recall that by $A_{M^{\text{HALT}},w,k}$ we mean the true sequence of oracle responses that would be seen by M^{HALT} if we were to run it for k steps on input w. We make two observations:

(1) If M^{HALT} halts on w within k steps, then

$$A_{M^{\text{HALT}},w,k} = A_{M^{\text{HALT}},w,k+1} = \ldots = A_{M^{\text{HALT}},w,k+n}, n \in \mathbb{N}.$$

(2) Likewise, if M^{HALT} halts on w within k steps, then 2.1 implies that if we were to run N on input $\langle k, A_{M^{\text{HALT}},w,k} \rangle$, the simulated M' would also halt.

To be exact, we can show using induction that in any step i of the computation, the configuration C'_i of M' matches the configuration C_i of M^{HALT}, and that in addition the value of oracle-count matches the number of oracle calls made by M^{HALT}.

Moreover, its computation history would match that of the true M^{HALT}, and so in particular the number of oracle calls that it would make would be the same. I.e., it would use exactly $|A_{M^{\text{HALT}},w,k}|$ oracle calls, and so N would *not* halt and accept in line 15.

We argue that $\langle M^{\text{HALT}}, w \rangle \in \overline{\text{HALT}}^{\text{HALT}}$ iff $\langle N \rangle \in \text{ALL}_{TM}$:

(or, equivalently, that $\langle M^{\text{HALT}}, w \rangle \in \text{HALT}^{\text{HALT}}$ iff $\langle N \rangle \in \overline{\text{ALL}}_{TM}$)

Suppose that $\langle M^{\text{HALT}}, w \rangle \in \text{HALT}^{\text{HALT}}$:

Then M^{HALT} halts on w in (exactly) k_0 steps for some $k_0 \in \mathbb{N}$. Let $A = A_{M^{\text{HALT}},w,k}$. Note by observation (1) that $A_{M^{\text{HALT}},w,k+n} = A$ for any $n \in \mathbb{N}$.

Now, suppose that we run N on $\langle k, A \rangle$ for $k \geqslant k_0$.

By observation (2) we see that the simulated computation history for M' will match the true computation history of M^{HALT}, and so in particular M' will halt and use exactly $|A|$ oracle calls. So N will not halt and accept before line 16.

In fact, since the computation history of M' matches that of M^{HALT}, the contents of its oracle tape during its oracle calls must match those of M^{HALT}.

Now, we have appended these tape contents to Φ^+ or Φ^- according to the values of $A[$*oracle-count*$]$, which we have argued match the true oracle responses. So Φ^+ contains exactly the TM/input pairs $\langle M_i, w_i \rangle$ for which M_i halts on w_i, and Φ^- contains exactly the TM/input pairs $\langle M_i, w_i \rangle$ for which M_i loops on w_i.

Since, for each $\langle M_i, w_i \rangle$ in Φ^+, M_i halts on w_i, there must be some $k_1 \in \mathbb{N}$ such that each M_i halts on w_i within k_1 steps (there are a finite number of pairs in Φ^+, so we can just take the maximum of the number of steps each takes to halt).

So if $k \geqslant \max(k_0, k_1)$, N will pass the loop on lines 16. to 18. without accepting, and so will move on to the loop on lines 19. to 22.

Since we have argued that for each TM/input pair $\langle M_i, w_i \rangle$, M_i loops on w_i, we can see that N will never reach an accepting configuration inside of this loop, and so if Φ^- contains at least one TM/input pair, then N will never finish the loop on lines 19. to 22. If there are no such TM/input pairs, then N will loop on line 23. Either way, N loops on the input $\langle k, A \rangle$.

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

Suppose that $\langle M^{\mathrm{HALT}}, w \rangle \in \overline{\mathrm{HALT}}^{\mathrm{HALT}}$:

Then M^{HALT} will not halt on w within k steps for any $k \in \mathbb{N}$.

So if we run N on any input $\langle k, A \rangle$, we have two possibilities:

- M' does not halt on w within k steps, or
- M' does halt on w within k steps, but A does not match the true oracle response string $A_{M^{\mathrm{HALT}},w,k}$.

Let's consider these possibilities.

- Now, if we run N on $\langle k, A \rangle$ and M' does not halt within k steps, then N will halt and accept in line 15.

On the other hand, suppose that we have some A such that M' does halt within k steps when run on $\langle k, A \rangle$. Then, as we have said, $A \neq A_{M^{\mathrm{HALT}},w,k}$.

- If A is a proper prefix of $A_{M^{\mathrm{HALT}},w,k}$ (so that $|A| < |A_{M^{\mathrm{HALT}},w,k}|$), then the computation history of M' in N will match that of M^{HALT} up until the $(|A|+1)^{st}$ oracel call, by 2.1. But on the $(|A|+1)^{st}$ oracel call N will halt and accept on line 7.
- Likewise, $A_{M^{\mathrm{HALT}},w,k}$ is a proper prefix of A, then 2.1 tells us that the computation history of M' will match that of M^{HALT} up to step k of the simulation, at which point N will halt and accept on line 15.

So the only possibility that concerns us is the case where M' halts on w within k steps for some $A \neq A_{M^{\mathrm{HALT}},w,k}$ such that $|A| = |A_{M^{\mathrm{HALT}},w,k}|$.

- Suppose that M' halts on w within k steps for some $A \neq A_{M^{\mathrm{HALT}},w,k}$ such that $|A| = |A_{M^{\mathrm{HALT}},w,k}|$. Then there must be some j such that $A[j] = 1$ and $A_{M^{\mathrm{HALT}},w,k}[j] = 0$, or there must be some j such that $A[j] = 0$ and $A_{M^{\mathrm{HALT}},w,k}[j] = 1$.

    If there is some j such that $A[j] = 1$ and $A_{M^{\mathrm{HALT}},w,k}[j] = 0$, then there is some TM/input pair $\langle M_j, w_j \rangle$ in Φ^+ such that M_j loops on w_j. In this case we can see that N will always accept on line 18. when considering this $\langle M_j, w_j \rangle$.

    If there is no j such that $A[j] = 1$ and $A_{M^{\mathrm{HALT}},w,k}[j] = 0$, but there is some j such that $A[j] = 0$ and $A_{M^{\mathrm{HALT}},w,k}[j] = 1$, then there is some TM/input pair $\langle M_j, w_j \rangle$ in Φ^- such that M_j halts on w_j within k_j steps for some $k_j \in \mathbb{N}$. But then N will pass into the loop from lines 19. to 22., and after running this loop for k_j iterations, M_j will halt on w_j and N will halt and accept.

  Since every input to N must satisfy one of these possibilities, we can see that N must accept every input.

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

Therefore we can conclude that

1) $\mathrm{ALL}_{\mathrm{TM}} \in \Pi_2$ and that
2) For any $L \in \Pi_2$, $L \leqslant_m \mathrm{ALL}_{\mathrm{TM}}$, as we we have claimed.

$\blacksquare$

5 Post's Theorem: Quantifiers and the Arithmentic Hierarchy

We've defined the Arithmetic Hierarchy by using oracles, but there are alternative definitions and constructions as well – and in particular, the classes in the Arithmetic Hierarchy can be characterized in way that only uses TMs – while oracles are useful as analytic tools, they are at the end of the day, fictional. But the classes we see here are meaningful even without that fiction.

Another way of describing language classes that we've run into in this course is through the use of quantifiers. Let's see if we can apply that sort of description of other levels of the Hierarchy. As a review:

- The recognizble languages (the class Δ_1) are the languages L such that there is some decidable predicate P for which
$$L = \left\{\langle x\rangle \,\middle|\, \exists y, P(x,y)\right\}.$$
- The co-recognizble languages (the class Π_1) are the languages L such that there is some decidable predicate Q for which
$$L = \left\{\langle x\rangle \,\middle|\, \forall y, P(x,y)\right\}.$$
- The decidable languages (the class Δ_1) are the languages that are both Σ_1 and Π_1.

We can extend this chracterization to the rest of the Arithmetic Hierarchy, as well:

- Suppose that $L \in \Sigma_2$. Then L is recognizable using some HALT OTM R^{HALT}. So we can characterize L as follows:
$$L = \left\{\langle x\rangle \,\middle|\, \exists k \in \mathbb{N}, R^{\mathrm{HALT}} \text{ accepts } x \text{ within } k \text{ steps.}\right\}$$

This is in set notation, but the predicate we're using isn't decidable. let's try to expand it in order to get a decidable predicate.

We can follow the same basic process as we did in our $\overline{\mathrm{HALT}^{\mathrm{HALT}}} \leqslant_m \mathrm{ALL}_{\mathrm{TM}}$ reduction: we'll get rid of the OTM by replacing it with a simulted OTM $\hat{R}$ that takes out the oracle and uses an oracle certificate A in its place *(we'll say that if A is the wrong length, if it does not match the number of oracle calls made in the first k steps of $\hat{R}$, then $\hat{R}$ loops forever).* If we do so then we can write L as

$$L = \left\{ \langle x \rangle \;\middle|\; \begin{array}{c} \exists k \in \mathbb{N}, A \in \{0,1\}^*, (\hat{R} \text{ accepts } x \text{ within } k \text{ steps using the oracle certificate } A) \\ \wedge\ (A \text{ is the correct oracle certificate}). \end{array} \right\}$$

The first of the clauses in this predicate – the claim that

$$\hat{R} \text{ accepts } x \text{ within } k \text{ steps using the oracle certificate } A$$

is decidable. So let's expand the second clause:

$$A \text{ is the correct oracle certificate.}$$

When we run $\hat{R}$ on x for k steps with the oracle certificte A, it will use A to split the oracle calls into its claimed *yes* and *no* instances $\Phi^+_{\hat{R},x,k}$ and $\Phi^-_{\hat{R},x,k}$. Note that the oracle certificate A is correct iff these $\Phi^+_{\hat{R},x,k}$ and $\Phi^-_{\hat{R},x,k}$ are also correct – that is, iff $\Phi^+_{\hat{R},x,k}$ is a list of *yes*-instances to HALT and $\Phi^-_{\hat{R},x,k}$ is a list of *no*-instnaces of HALT.

Now, suppose we write a TM M_H that takes as its input a list $\langle T \rangle$ if TM/input pairs:

```
Let $M_H$ = "On input $\langle T \rangle$:
  1. For $\langle M, w \rangle \in T$:
  2.     Run $M$ on $w$.
  3. *Accept*."
```

Clearly, M_H halts iff every $\langle M, w \rangle$ in its input halts. So $\Phi^+_{\hat{R},x,k}$ is a list of *yes*-instances to HALT iff M_H halts when run on $\Phi^+_{\hat{R},x,k}$ – i.e., iff there is some $k' \in \mathbb{N}$ such that $\langle M_H, \Phi^+_{\hat{R},x,k}, k' \rangle \in \mathrm{H}$.

Similarly, we can write a TM M_L that also takes as its input a list $\langle T \rangle$ if TM/input pairs:

```
Let $M_L$ = "On input $\langle T \rangle$:
  1. For $i = 0$ to $\infty$
  2.     For each $\langle M, w \rangle \in T$:
  3.         Run $M$ on $w$ for $i$ steps.
  4.         If ny halt, *accept*."
```

Clearly, M_L halts iff any $\langle M, w \rangle$ in its input halts. So $\Phi^-_{\hat{R},x,k}$ is a list of *no*-instances to HALT iff M_H loops when run on $\Phi^-_{\hat{R},x,k}$ – i.e., iff for all $k'' \in \mathbb{N}$, $\langle M_L, \Phi^-_{\hat{R},x,k}, k'' \rangle \notin \mathrm{H}$.
We can therefore expand our earlier clause

$$A \text{ is the correct oracle certificate.}$$

into the form

$$\exists k', \forall k'', \langle M_H, \Phi^+_{\hat{R},x,k}, k' \rangle \in \mathrm{H} \wedge \langle M_L, \Phi^-_{\hat{R},x,k}, k'' \rangle \notin \mathrm{H}.$$

This clause uses only quantifiers and decidable predicates, and when we plug it into our description of the language L we get:

$$L = \left\{ \langle x \rangle \,\middle|\, \exists k \in \mathbb{N}, R^{\mathrm{HALT}} \text{ accepts } x \text{ within } k \text{ steps.} \right\}$$

$$= \left\{ \langle x \rangle \,\middle|\, \begin{array}{c} \exists k \in \mathbb{N}, A \in \{0,1\}^*, (\hat{R} \text{ accepts } x \text{ within } k \text{ steps using the oracle certificate } A) \\ \land\ (A \text{ is the correct oracle certificate}) \end{array} \right\}$$

$$= \left\{ \langle x \rangle \,\middle|\, \begin{array}{l} \exists k, k' \in \mathbb{N}, A \in \{0,1\}^*, \forall k'' \in \mathbb{N}, \\ \qquad\qquad (\hat{R} \text{ accepts } x \text{ within } k \text{ steps using the oracle certificate } A) \\ \qquad \land\ (\langle M_H, \Phi^+_{\hat{R},x,k}, k' \rangle \in \mathrm{H}) \\ \qquad \land\ \langle M_L, \Phi^-_{\hat{R},x,k}, k'' \rangle \notin \mathrm{H} \end{array} \right\}.$$

So for any $L \in \Sigma_2$ then there is some decidable P such that we can write L in the form

$$L = \left\{ \langle x \rangle \,\middle|\, \exists y, \forall z, P(x,y,z) \right\}.$$

In fact, the converse holds as well: if we have a decidable P such that we can write L in the form

$$L = \left\{ \langle x \rangle \,\middle|\, \exists y, \forall z, P(x,y,z) \right\}.$$

Now, consider the following HALT-OTM:

```
Let $y_0, y_1, \ldots$ be an effective enumeration of the $y$.
Let $z_0, z_1, \ldots$ be an effective enumeration of the $z$.
Let $R^{\mathrm{HALT}}$ = "On input $\langle x \rangle$:
  1. For $i = 0$ to $\infty$:
  2.     Let $M_{y_i}$ = "On input $w$:
             1. For $j = 0$ to $\infty$:
             2.     If $P(x, y_i, z_j)$:
             3.         Accept.
             4. Loop.
  3.     Query the oracle on $\langle M_i, \varepsilon \rangle$.
  4.     If it says no, then accept."
```

We can see that this OTM recognizes L. So any L of the form

$$L = \left\{ \langle x \rangle \,\middle|\, \exists y, \forall z, P(x,y,z) \right\}.$$

is an element of Σ_2.

What this means:

- We can characterize Σ_2 as precicely the class of languages L such that there is some decidable P for which

$$L = \big\{\langle x\rangle \,\big|\, \exists y, \forall z, P(x,y,z)\big\}.$$

- Since Π_2 is made up of the L such that $\overline{L} \in \Sigma_2$, we can see that Π_2 is precisely the class of L such that there is some decidable Q for which

$$L = \big\{\langle x\rangle \,\big|\, \forall y, \exists z, Q(x,y,z)\big\}.$$

- Since $\Delta_2 - \Sigma_2 \cap \Pi_2$, the class Δ_2 is precisely the class of L such there there are decidable predicates P and Q for which

$$\begin{aligned} L &= \big\{\langle x\rangle \,\big|\, \exists y, \forall z, P(x,y,z)\big\} \\ &= \big\{\langle x\rangle \,\big|\, \forall y', \exists z', Q(x,y',z')\big\} \end{aligned}$$

In fact, a similar characterization can be made for each level of the Arithmentic Hierarchy – we call this result *Post's Theorem*:

Theorem 5.1 (Post's Theorem) *Let $L \in \Sigma_i$. Then there is some decidable predicate P such that*

$$L = \big\{\langle x\rangle \,\big|\, \Box_1 y_1, \Box_2 y_2, \ldots, \Box_i y_i P(x, y_1, y_2, \ldots, y_i)\big\},$$

where

$$\Box i = \begin{cases} \exists, & i \text{ is odd,} \\ \forall, & i \text{ is even.} \end{cases}$$

Similarly, let $L' \in \Pi_i$. Then there is some decidable Q such that

$$L' = \big\{\langle x\rangle \,\big|\, \Box_1 y_1, \Box_2 y_2, \ldots, \Box_i y_i Q(x, y_1, y_2, \ldots, y_i)\big\},$$

where

$$\Box i = \begin{cases} \exists, & i \text{ is even,} \\ \forall, & i \text{ is odd.} \end{cases}$$

A language L is in Δ_i iff it is in $\Sigma_i \cap \Pi_i$, in which case it can be written in both of these forms.

While we won't give a full (inductive) proof of this result here, we will note that the base case for Δ_1, Σ_1, and Π_1 follows from what we have done in class, and that the inductive steps are similar to what we have done above for Σ_2.