Arithmetical Hierarchy Notes

Ziv Scully

CMU 15-455

1 What is the Arithmetical Hierarchy?

We are familiar with classifying problems into decidable and undecidable. The arithmetical hierarchy is a way of classifying problems into not just decidable and undecidable but, in the latter case, *how far from decidable* a problem is. For example, problems that are semidecidable or cosemidecidable are in some sense closer to decidable than problems that aren't.

As a running example, we'll keep returning to the following problems:

$$\begin{aligned}\mathsf{Halts}(e, x) &= \{e\}(x) \text{ halts},\\ \mathsf{Total}(e) &= \{e\} \text{ halts on every input.}\end{aligned}$$

Both Halts and Total are undecidable, but intuitively, Total seems like a harder property to show. We will see later that Halts and Total are classified differently by the arithmetical hierarchy.

1.1 Clarification: Languages and Predicates

Before we begin, let's take a second to clarify what "problems" means. In this document, we'll talk about classifying two related but slightly different types of things:

- *languages*, which are sets of strings (or tuples of strings); and
- *predicates*, which are logical statements parametrized by a string (or tuple of strings).

It is easy to convert between languages and predicates. For example, given a predicate P, the corresponding language L_P is

$$L_P = \{x \mid P(x)\},$$

and given a language L, the corresponding predicate P_L is

$$P_L(x) = x \in L.$$

Finally, because I'm writing this document the day before the midterm and thus have limited time, we'll play a little fast and loose with the distinction between strings and tuples of strings.

We've mostly focused on languages in this course, but for this document it will be easier to think in terms of predicates. For example, Halts and Total above are predicates.

1.2 Halting on One Input vs. Halting on Every Input

Before formally defining the arithmetical hierarchy, let's try to get a sense of "how undecidable" each of Halts and Total is.

We start with Halts. Recall that $\mathsf{Halts}(e, x)$ is true when $\{e\}(x)$ halts. While Halts is undecidable, a closely related predicate is decidable:[1]

$$\mathsf{HaltsWithin}(e, x, \sigma) = \{e\}(x) \text{ halts within } \sigma \text{ steps.}$$

[1] In class, we've expressed $\mathsf{HaltsWithin}(e, x, \sigma)$ as $x \in W_{e,\sigma}$, where $W_{e,\sigma}$ is the set of inputs for which $\{e\}$ halts within σ steps.

To measure “how undecidable” Halts is, we see how much we need to add to the decidable HaltsWithin to express Halts. We can write Halts in terms of HaltsWithin as follows:

$$\mathsf{Halts}(e, x) = \exists\sigma, \mathsf{HaltsWithin}(e, x, \sigma).$$

Roughly speaking, Halts is “one $\exists$ away” from HaltsWithin. This makes Halts semidecidable: to semidecide $\mathsf{Halts}(e, x)$, we can search for a σ such that $\mathsf{HaltsWithin}(e, x, \sigma)$ holds. If we find such a σ, we can halt and say $\mathsf{Halts}(e, x)$ is true, but if we never find such a σ, we will search forever without halting.

We now turn to Total. Recall that $\mathsf{Total}(e)$ is true when $\{e\}$ halts on every input x. This means

$$\mathsf{Total}(e) = \forall x, \mathsf{Halts}(e, x) = \forall x, \exists\sigma, \mathsf{HaltsWithin}(e, x, \sigma).$$

Roughly speaking, Total is “a $\forall$ and an $\exists$ away” from HaltsWithin, which is even further away than Halts. It turns out that Total is neither semidecidable nor cosemidecidable, so it really is harder than Halts. Here’s the intuition for why:

- To semidecide Total, we would need to write a program that, given input e, halts if $\mathsf{Total}(e)$ is true. For any one input x, it’s possible to semidecide $\mathsf{Halts}(e, x)$. However, we need to try this for all inputs x, and for any finite number of inputs we try, we will always be worried that some future input will cause our machine not to halt.
- To cosemidecide Total, we would need to write a program that, given input e, halts if $\mathsf{Total}(e)$ is false. It suffices to search for one input x such that $\mathsf{Halts}(e, x)$ is false. Unfortunately, there is no program that halts whenever $\mathsf{Halts}(e, x)$ is false.[2]

1.3 Defining the Arithmetical Hierarchy

The arithmetical hierchy is a collection of *sets of predicates*. These sets of predicates come in three flavors:

- $\Sigma_0, \Sigma_1, \ldots$;
- $\Pi_0, \Pi_1, \ldots$; and
- $\Delta_0, \Delta_1, \ldots$.

The collection of these three sequences of sets constitutes the *arithmetical hierarchy*.

The arithmetical hierarchy is defined inductively as follows. We start by defining Σ_0 and Π_0:

$$\Sigma_0 = \Pi_0 = \{\text{predicates } P \mid P \text{ is decidable}\}. \tag{1.1}$$

Then, for all $i \geq 1$, we define Σ_i and Π_i in terms of Σ_{i-1} and Π_{i-1}:

$$\begin{aligned}\Sigma_i &= \{\text{predicates } P \mid P(x) \text{ is equivalent to } \exists y, Q(x, y) \text{ for some } Q \in \Pi_{i-1}\},\\ \Pi_i &= \{\text{predicates } P \mid P(x) \text{ is equivalent to } \forall y, Q(x, y) \text{ for some } Q \in \Sigma_{i-1}\}.\end{aligned}$$

Finally, for all $i \geq 0$, we define Δ_i:

$$\Delta_i = \Sigma_i \cap \Pi_i.$$

Some notes about the definitions:

- The arithmetical hierarchy is, well, a nested hierarchy of sets. Specifically, for all $i \geq 1$,

  $$\Sigma_{i-1} \subseteq \Delta_i, \qquad \Pi_{i-1} \subseteq \Delta_i, \qquad \Delta_i \subseteq \Sigma_i, \qquad \Delta_i \subseteq \Pi_i.$$

  There are nice pictures of the hierarchy on the lecture slides.

[2]This is because if there were such a program, we could write a decider for Halts, but we know Halts is undecidable.

- By recursively expanding the definitions of Σ_i and Π_i, one can characterize them by *the number of alternating quantifiers* outside of a core decidable predicate:

$$\Sigma_i = \left\{ \text{predicates } P \;\middle|\; \begin{array}{l} P(x) \text{ is equivalent to} \\ \exists y_1, \forall y_2, \exists y_3, \forall y_4, \ldots, [\exists \text{ or } \forall] y_n, Q(x, y_1, \ldots, y_n) \\ \text{for some decidable predicate } Q \end{array} \right\},$$

$$\Pi_i = \left\{ \text{predicates } P \;\middle|\; \begin{array}{l} P(x) \text{ is equivalent to} \\ \forall y_1, \exists y_2, \forall y_3, \exists y_4, \ldots, [\exists \text{ or } \forall] y_n, Q(x, y_1, \ldots, y_n) \\ \text{for some decidable predicate } Q \end{array} \right\}.$$

For example,

$$\Sigma_2 = \{\text{predicates } P \mid P(x) \text{ is equivalent to } \exists y_1, \forall y_2, Q(x, y_1, y_2) \text{ for some decidable predicate } Q\}.$$

This "count the alternating quantifiers" view is one of the most important ways to view the arithmetical hierarchy.
- It is possible to squish multiple quantifiers of the same type together. For example, $\exists y_1, \exists y_2, Q(x, y_1, y_2)$ is equivalent to $\exists (y_1, y_2), Q(x, y_1, y_2)$, and similar for $\forall$. This is why we can work just alternating quantifiers, as we can squish repetitions together.
- What about Δ_i? How do we view that in terms of quantifiers? The way to think about Δ_i is that a predicate $P \in \Delta_i$ can be written using n alternating quantifiers around a core decidable predicate, *but we can start with either* $\exists$ *or* $\forall$.
- As a special case, it turns out $\Delta_1 = \Sigma_0 = \Pi_0$, namely Δ_1 contains the decidable predicates. Exercise: if a predicate is in both Σ_1 and Π_1, why is it decidable?

Let's see how our running examples fit into the arithmetical hierarchy. We have previously written Halts as (see Section 1.2)

$$\mathsf{Halts}(e, x) = \exists \sigma, \mathsf{HaltsWithin}(e, x, \sigma).$$

The predicate HaltsWithin is decidable, so we can write Halts as a decidable core predicate surrounded by a single $\exists$, so Halts $\in \Sigma_1$.

Similarly, we have previously expressed Total as (see Section 1.2)

$$\mathsf{Total}(e) = \forall x, \exists \sigma, \mathsf{HaltsWithin}(e, x, \sigma).$$

The predicate HaltsWithin is decidable, so we can write Halts as a decidable core predicate surrounded by two alternating quantifiers starting with $\forall$, so Total $\in \Pi_2$.

2 How to Solve Arithmetical Hierarchy Problems

Before covering how to solve arithmetical hierarchy problems, let us first discuss what such problems look like. For the most part, whenever you see an arithmetical hierarchy problem in this class, it will be of the form "find the location of P in the arithmetical hierarchy", where P is some predicate (or language; see Section 1.1). The problem will usually say "focus on the upper bound and don't worry about the lower bound". What this question means is to find the smallest set in the arithmetical hierarchy that you can prove contains predicate P. Specifically, recalling (1.1), the nesting of the arithmetical hierarchy is as follows:

- $\Delta_0 = \Sigma_0 = \Pi_0 = \Delta_1$ is the set of decidable predicates. It is a subset of...
- ...both Σ_1 and Π_1, neither of which is a subset of the other. But both Σ_1 and Π_1 are subsets of ...

- …Δ_2, which is a subset of both …
- …Σ_2 and Π_2, which are both subsets of
- Δ_3, etc.

So when we ask you to find an upper bound for a predicate P, we're asking you to find the Σ_i, Π_i, or Δ_i set containing P that appears as early in the list above as you can manage. There will never be a tie, because if a predicate is in both Σ_i and Π_i, then it is in $\Delta_i = \Sigma_i \cap \Pi_i$, which appears earlier in the list. We are *not* asking you to prove that you found the tightest possible bound, just to try your best.

As for how to solve the problems: the key is to break the statement down into decidable pieces. I suggest always expanding out all the quantifiers you can until your left with basic logic (and, or, not) and decidable predicates like HaltsWithin. Then worry about rearranging the formula so that all the predicates are on the outside. For a good example of this, look at the solutions to Homework 2, Question 4, or at the Midterm Practice Problems, Question 2.