The arithmetic hierarchy

Vassos Hadzilacos[1]

1 Definition and basic properties

The arithmetic hierarchy is a classification of some languages based on their "degree of uncomputability": At the bottom of the hierarchy lie the decidable languages, contained by the larger sets of recognizable and co-recognizable languages. These, in turn, are contained by a larger set of undecidable languages and their complements; and these sets are contained by a larger set of languages that contain "even more undecidable" languages and their complements; and so on *ad infinitum*. Each such pair of sets defines a "level" of this infinite hierarchy of more and more undecidable languages. Remarkably, there are two quite different, but equivalent, ways of arriving at the definition of these sets of languages: One is based on oracle Turing machines (OTMs) and the other is based on the structure of first-order logic formulas that can be used to describe them.

Note that OTMs can be encoded by strings in a way that is similar to how ordinary Turing machines are. Furthermore, there is a Universal OTM M_U which, given the code $\langle M, x\rangle$ of an OTM M and input x, uses oracle A to simulate the computation of M^A on x. The Universal OTM works just like the Universal Turing machine, and uses its oracle for the query steps. (For more information on OTMs see the document entitled "Reductions", specifically the discussion of Turing reductions; and for more information on the Universal Turing machine see the document entitled "Church's thesis and the Universal Turing machine.)

Definition 1 *For every $i \in \mathbb{N}$, we define the sets Δ_i, Σ_i, and Π_i inductively as follows:*

- $\Delta_0 = \Sigma_0 = \Pi_0 =$ *the set of decidable languages.*
- *For each $i > 0$,*
  - *Δ_i is the set of languages that are **decided** by OTMs that use oracles in Σ_{i-1}.*
  - *Σ_i is the set of languages that are **recognized** by OTMs that use oracles in Σ_{i-1}; and Π_i is the set of complements of languages in Σ_i.*

*The sets Δ_i, Σ_i, and Π_i comprise the i-th level of the **arithmetic hierarchy**.*

A decidable language is of no help as an oracle, since membership in it can be decided by a Turing machine, without needing to resort to an oracle. Therefore, the following important fact about level 1 of the arithmetic hierarchy is an immediate corollary of Definition 1.

Corollary 2 *Δ_1 is the set of decidable languages, Σ_1 is the set of recognizable languages, and Π_1 is the set of co-recognizable languages.*

We have seen that the set of decidable languages, Δ_1, is closed under complementation. This is also true for every Δ_i for essentially the same reason: By swapping the accept and reject states of an OTM that decides L using L', we obtain an OTM that decides $\overline{L}$ using L'.

[1]I am grateful to Yiyang Zhou, former teaching assistant for this course, who provided extensive comments and corrections that improved this document significantly. Naturally I am responsible for any remaining errors.

Corollary 3 *For each $i \in \mathbb{N}$, Δ_i is closed under complementation.*

The next result states that each level of the arithmetic hierarchy is contained in the next.

Theorem 4 *For each integer $i \geq 1$, (a) $\Delta_i \subseteq \Sigma_i \subseteq \Delta_{i+1}$, and (b) $\Delta_i \subseteq \Pi_i \subseteq \Delta_{i+1}$.*

PROOF. Let $i \geq 1$.

For part (a), the fact that $\Delta_i \subseteq \Sigma_i$ follows immediately from the fact that a decider that uses L is, *a fortiori*, a recognizer that uses L. To show that $\Sigma_i \subseteq \Delta_{i+1}$, consider any language $L \in \Sigma_i$. We define an OTM M such that M^L decides L: M simply copies its input x to the query tape, enters the query state, and accepts if the response to the query is 1 and rejects if the response is 0. Clearly M^L decides L, and since $L \in \Sigma_i$, by definition $L \in \Delta_{i+1}$. Thus, $\Sigma_i \subseteq \Delta_{i+1}$.

For part (b), let $L \in \Delta_i$. By Corollary 3 $\overline{L} \in \Delta_i$ and by part (a) $\overline{L} \in \Sigma_i$ and so $L \in \Pi_i$, proving $\Delta_i \subseteq \Pi_i$. Now let $L \in \Pi_i$ and so $\overline{L} \in \Sigma_i$; by part (a) $\overline{L} \in \Delta_{i+1}$, and by Corollary 3 $L \in \Delta_{i+1}$, proving that $\Pi_i \subseteq \Delta_{i+1}$. Thus $\Delta_i \subseteq \Pi_i \subseteq \Delta_{i+1}$. □

Are the containments in Theorem 4 proper? That is, is it the case that for all $i \geq 1$, $\Delta_i \subsetneq \Sigma_i \subsetneq \Delta_{i+1}$, and $\Delta_i \subsetneq \Pi_i \subsetneq \Delta_{i+1}$? It is conceivable that once we give an OTM a powerful enough oracle in Σ_1, say an oracle that solves the Halting problem for Turing machines, the OTM becomes all-powerful, in the sense that it can solve all problems solvable by OTMs and the hierarchy "collapses" — there is only our familiar level 1 and then all other problems in a second level. As we will prove next, this is not the case: Although an OTM with an oracle for the Halting problem for TMs can trivially solve the Halting problem for TMs (and therefore can decide any recognizable language) it cannot solve the Halting problem for OTMs with an oracle for the Halting problem for TMs! This "jump" in difficulty occurs at every level and so the containments of Theorem 4 are indeed proper: The arithmetic hierarchy has an infinite number of distinct levels, each properly contained in the next. Figure 1 illustrates the structure of the arithmetic hierarchy as a Venn diagram of the sets involved. The languages H^i, $\overline{H^i}$, and D^i that "separate" each level i from the previous one will be defined shortly, as we develop the technical machinery needed to prove that the containments of Theorem 4 are indeed proper.

Definition 5 *The **Halting problem with oracle** L is the problem of determining whether a given OTM N using oracle L halts on a given input x; that is, it is the language*

$$H^L = \{\langle N, x\rangle : \ N \text{ is an OTM such that } N^L \text{ halts on input } x\}.$$

Theorem 6 *For every language L, the language*

$$D^L = \{\langle N\rangle : \ N \text{ is an OTM such that } N^L \text{ does not halt on input } \langle N\rangle\}$$

is not recognizable by any OTM using L.

PROOF. Let L be any language and suppose, for contradiction, that there is an OTM M such that M^L recognizes D^L. Without loss of generality, assume that M never rejects — it loops instead. Thus, for any input $\langle N\rangle$,

$$M^L \text{ halts on } \langle N\rangle \quad \Leftrightarrow \quad M^L \text{ accepts } \langle N\rangle \quad \Leftrightarrow \quad N^L \text{ does not halt on } \langle N\rangle.$$

In particular, taking $N = M$, i.e., running M with oracle L on its own code, we get

$$M^L \text{ halts on } \langle M\rangle \quad \Leftrightarrow \quad M^L \text{ does not halt on } \langle M\rangle,$$

which is a contradiction. □

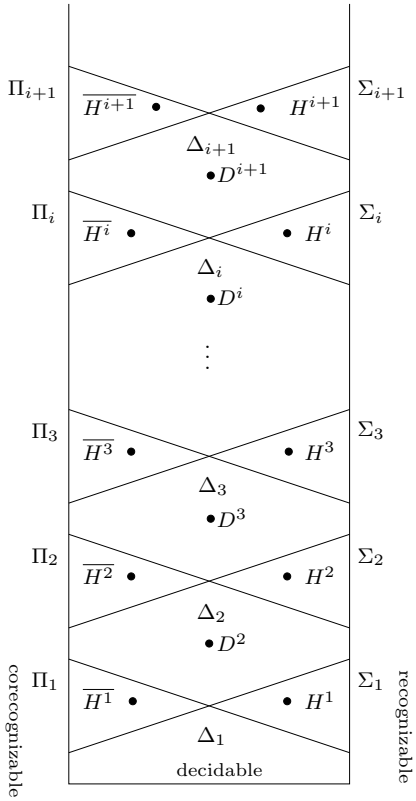

Figure 1: Structure of the arithmetic hierarchy

Theorem 7 *For every language L, H^L is (a) recognizable by some OTM that uses L, but (b) not decidable by any OTM that uses L.*

PROOF. The following is a recognizer for H^L: on input $\langle M, x\rangle$, run M using L on input x. If this computation halts, then accept. This proves part (a).

For part (b) suppose, for contradiction, that there is some language L and some OTM M such that M^L decides H^L. Then the language D^L is also decidable by the following OTM using L: On input $\langle N\rangle$, run M using L on input $\langle N, N\rangle$ and do the opposite of what M does on that input: accept if M rejects and reject if M accepts. This contradicts Theorem 6. □

Notice that the set H is a language (the halting problem for Turing machines with no oracle), so H^H is also a language (the halting problem for OTMs with oracle H), and so $H^{(H^H)}$ is also a language (the halting problem for OTMs with oracle H^H), and so on *ad infinitum*. We use the notation H^i to denote the language resulting from iterating this construction i times. More precisely,

Definition 8 *For any $i \in \mathbb{N}$, define H^i inductively as follows:*

- $H^0 = \{\epsilon\}$ [2]
- *for $i > 0$, $H^i = H^{H^{i-1}}$ (the halting problem for OTMs using H^{i-1} as the oracle).*

Note that H^1 is the usual halting problem for Turing machines (with no oracle).

[2]We could use any decidable set as H^0 — except, for somewhat technical reasons, $\varnothing$ and Σ^* — so we fix a specific very simple decidable set other than these two.

Next we define the important concept of a language being ***complete*** for a set of languages $\mathcal{L}$. A variant of this concept plays a key role in the second part of the course, when we discuss complexity theory.

Definition 9 *Let $\mathcal{L}$ be a set of languages, and L be a language. We say that L is **complete for** $\mathcal{L}$ (or **$\mathcal{L}$-complete**) if*

(a) $L \in \mathcal{L}$, and

(b) for every L' in $\mathcal{L}$, $L' \leq_m L$.

Thus, a language is $\mathcal{L}$-complete if it is one of the "hardest" languages in $\mathcal{L}$: If we can decide it, the we can decide ***all*** languages in $\mathcal{L}$.[3]

Lemma 10 *Let L be a language that is recognized (respectively, decided) by an OTM M_L using oracle A, and let A' be such that $A \leq_m A'$. Then L is also recognized (respectively, decided) by an OTM N_L using oracle A'.*

Proof. Intuitively this is clear because $A \leq_m A'$ means that A' is at least as strong as A, so if something can be done using A it can also be done using A'. More precisely we argue as follows.

Let f be a mapping reduction of A to A'. Let M_L be an OTM that uses A to recognize (respectively, decide) L. We define an OTM N_L that works like M except that, when M_L is about to ask its oracle the query "$x \in A$?", OTM N_L first computes $f(x)$ (which it can do because f, being a mapping reduction, is computable) and then asks its oracle the query "$f(x) \in A'$?". The two queries have identical answers (because $x \in A$ if and only if $f(x) \in A'$), so M_L^A on any input performs exactly the same computation as $N_L^{A'}$ on the same input. Therefore $N_L^{A'}$ recognizes (respectively, decides) L, just as M_L^A does. □

Theorem 11 (Post) *For every $i \in \mathbb{N}$, H^i is complete for Σ_i.*

Proof. We use induction to prove that, for every $i \in \mathbb{N}$,

(a) $H^i \in \Sigma_i$, and

(b) for every $L \in \Sigma_i$, $L \leq_m H^i$.

For the basis $i = 0$, recall that $H^0 = \{\epsilon\}$ and Σ_0 is the set of decidable languages, so (a) is obvious. For (b), let L be any language in Σ_0. Then consider the function f that maps every $x \in L$ to ϵ, and every $x \notin L$ to some string other than ϵ.[4] The function f is computable because L is decidable, and so there is a TM to decide whether $x \in L$.

For the induction step, let $i > 0$ and suppose the result holds for $i - 1$. By definition, $H^i = H^{H^{i-1}}$ and so, by Theorem 7(a), H^i is recognizable using H^{i-1}. Since, by the induction hypothesis, $H^{i-1} \in \Sigma_{i-1}$, it follows that $H^i \in \Sigma_i$, proving part (a) of the induction step. For part (b), let L be any language in Σ_i. We will prove that $L \leq_m H^i$.

Since $L \in \Sigma_i$, for $i > 0$, there is an OTM M_L and a language $A \in \Sigma_{i-1}$ such that M_L^A recognizes L. By the induction hypothesis, $A \leq_m H^{i-1}$. Thus, by Lemma 10, there is also an OTM N_L such that $N_L^{H^{i-1}}$ recognizes L. We now describe a mapping reduction of L to H^i. Given x, we construct the code of the following OTM M:

[3] There are, in fact, two kinds of completeness, mapping-completeness (based on mapping reductions $\leq_m$) and Turing-completeness (based on Turing reductions $\leq_T$). In this course we will deal exclusively with the more refined notion of completeness, based on mapping reductions.

[4] You can see now why we could not use $\varnothing$ or the set of all strings for H^0.

$M :=$ **on input** y:
1 run N_L on x (using M's oracle)
2 **if** N_L accepts x **then halt** (either accept or reject y, it doesn't matter which)
3 **else loop**

Using oracle H^{i-1}, this OTM halts on x if and only if $x \in L$:

- If $x \in L$ then line 1 terminates with $N_L^{H_{i-1}}$ having accepted x, and so M on any input (and in particular on input x) halts in line 2. Thus, in this case, $\langle M, x\rangle \in H^i$.
- If $x \notin L$ then $N_L^{H_{i-1}}$ either loops on x and so line 1 does not terminate, or it rejects x and so line 3 is executed. Either way $N_L^{H_{i-1}}$ loops on every input (and in particular on input x). Thus, in this case, $\langle M, x\rangle \notin H^i$.

Clearly the mapping $x \mapsto \langle M, x\rangle$ can be computed by a Turing machine, so this is a mapping reduction of L to H^i, as wanted. □

An immediate consequence of this and Lemma 10 is the following:

Corollary 12 *If $L \in \Sigma_i$ (respectively, $L \in \Delta_i$) then there is an OTM that recognizes (respectively, decides) L using H^{i-1}.*

Another immediate consequence of Theorem 11 is the following:

Corollary 13 *$\overline{H^i}$ is Π_i-complete.*

Definition 14 *For all integers $i \geq 1$, define*

$$D^i = \{\langle M, x, 0\rangle \colon \langle M, x\rangle \in H^{i-1}\} \cup \{\langle M, x, 1\rangle \colon \langle M, x\rangle \in \overline{H^{i-1}}\}$$

Intuitively, a string $\langle M, x, b\rangle$ in D^i tells us whether $\langle M, x\rangle \in H^{i-1}$ ($b = 0$) or not ($b = 1$).

We can now prove a number of useful facts about the arithmetic hierarchy, some of which generalize things we have proved about its first level, i.e., the set of decidable, recognizable, and co-recognizable languages.

Theorem 15 *For every integer $i \geq 1$,*

(a) $\Delta_i = \Sigma_i \cap \Pi_i$,

(b) $H^i, \overline{H^i} \in \Delta_{i+1} - \Delta_i$,

(c) $H^i \in \Sigma_i - \Pi_i$, *and*

(d) $\overline{H^i} \in \Pi_i - \Sigma_i$,

(e) $D^i \in \Delta_i$, *and*

(f) $D^{i+1} \notin \Sigma_i \cup \Pi_i$.

Proof.

(a) From Theorem 4 we have that $\Delta_i \subseteq \Sigma_i \cap \Pi_i$. For the reverse inclusion, consider any language $L \in \Sigma_i \cap \Pi_i$. Then there are OTMs M and $\overline{M}$ that recognize L and $\overline{L}$, respectively, using some oracles A and A' in Σ_{i-1}. By Corollary 12, there are OTMs N and $\overline{N}$ that recognize L and $\overline{L}$ using H^{i-1}. Then the following OTM using H^{i-1} decides L:

```
1  on input x:
2      for t := 1, 2, 3, ... do
3          run N on x for t steps or until it halts, whichever happens first
4          if N accepts x within t steps then accept x
5          run N̄ on x for t steps or until it halts, whichever happens first
6          if N̄ accepts x within t steps then reject x
```

Since every x belongs to either L or to $\overline{L}$, either $N^{H^{i-1}}$ accepts x within some number of steps, in which case $x \in L$, and the above OTM using H^{i-1} accepts x; or $\overline{N}^{H^{i-1}}$ accepts x within some number of steps, in which case $x \notin L$, and the above OTM using H^{i-1} rejects x. Therefore the above OTM using H^{i-1} decides L. By Theorem 11 $H^{i-1} \in \Sigma_{i-1}$. So, L is decidable by an OTM using an oracle in Σ_{i-1}; by definition of Δ_i, $L \in \Delta_i$.

(b) H^i and $\overline{H^i}$ are clearly both in Δ_{i+1} since they can be trivially decided using oracle H^i, which is in Σ_i by Theorem 11. To show that $H^i \notin \Delta_i$ suppose, for contradiction, that $H^i \in \Delta_i$. Then, by Corollary 12, there is an OTM M that decides H^i using oracle H^{i-1}. But recall that, for $i > 0$, $H^i = H^{H^{i-1}}$, so there is an OTM that decides $H^{H^{i-1}}$ using oracle H^{i-1}, which contradicts Theorem 7, thereby proving that $H^i \notin \Delta_i$. Since Δ_i is closed under complementation (Corollary 3), $\overline{H^i}$ is also not in Δ_i.

(c) The fact that $H^i \in \Sigma_i$ follows from Theorem 11. To show that $H^i \notin \Pi_i$, suppose for contradiction that $H^i \in \Pi_i$. So, by part (a), $H_i \in \Delta_i$, contradicting part (b).

(d) Follows directly from part (c), since each of Σ_i and Π_i contains the complements of the languages in the other.

(e) D^i is clearly decidable using oracle H^{i-1}, which is in Σ_{i-1} by part (c); therefore $D^i \in \Delta_i$.

(f) We have:

$$\begin{aligned} &H^i \leq_m D^{i+1} \text{ (via the mapping } \langle M, x\rangle \mapsto \langle M, x, 0\rangle) \quad \text{and} \\ &\overline{H^i} \leq_m D^{i+1} \text{ (via the mapping } \langle M, x\rangle \mapsto \langle M, x, 1\rangle) \end{aligned} \tag{1}$$

Suppose, for contradiction, that $D^{i+1} \in \Sigma_i \cup \Pi_i$. Since H^i is Σ_i-complete and $\overline{H^i}$ is Π_i-complete (Theorem 11 and Corollary 13), either $D^{i+1} \leq_m H_i$ or $D^{i+1} \leq_m \overline{H_i}$. Thus, by (1) and transitivity of mapping reductions, either $\overline{H^i} \leq_m H^i$ or $H^i \leq_m \overline{H^i}$, which imply, respectively, that either $\overline{H^i} \in \Sigma_i$ or $H^i \in \Pi_i$. These contradict parts (c) and (d). □

Theorem 16 *The inclusions in Theorem 4 are proper. That is, for each integer $i \geq 1$, (a) $\Delta_i \subsetneq \Sigma_i \subsetneq \Delta_{i+1}$, and (b) $\Delta_i \subsetneq \Pi_i \subsetneq \Delta_{i+1}$.*

Proof. This follows from Theorem 4 and the following facts:

- $H^i \in \Sigma_i - \Delta_i$ (Theorem 15(c) and (b)),
- $\overline{H^i} \in \Pi_i - \Delta_i$ (Theorem 15(d) and (b)),
- $D^{i+1} \in \Delta_{i+1} - (\Sigma_i \cup \Pi_i)$ (Theorem 15(e) and (f)). □

2 Alternative characterization of the arithmetic hierarchy

2.1 Predicates as languages

Let $R(x_1, \ldots, x_k)$ be a k-place predicate whose arguments $x_1, \ldots, x_k$ take on values from sets of finite mathematical objects, i.e., ones that can be encoded by strings over some alphabet. For example consider the predicate $T(M, x, c)$ defined as follows:

$T(M, x, c)$ is true if and only if c is an accepting computation of Turing machine M on input x.

This is known as ***Kleene's T-predicate***. We have seen that Turing machines and their inputs can be encoded by strings. An accepting computation c of a Turing machine M on input x can also be encoded by a string as follows: c is a finite sequence of configurations $C_0, C_1, \ldots, C_\ell$, where C_0 is the initial configuration q_0x of M on x, C_ℓ is an accepting configuration $y_1h_Ay_2$ of M, and for each $i \in [0..\ell - 1]$, $C_i \vdash_M C_{i+1}$ — i.e., each configuration is obtained from the previous one by a legal move of M. Each configuration is a finite string, and a finite sequence of strings can be encoded by a string (e.g., the concatenation of the configurations separated by a special symbol not occurring in the configurations).

Since the values of each argument of R can be encoded as strings over some alphabet, the values of R's k free variables can be encoded by a k-tuple of strings, and that k-tuple can be encoded by a single string over some alphabet. In this manner, R represents a language L_R, consisting of the set of encodings of k-tuples of values that satisfy the predicate: $L_R = \{\langle x_1, \ldots, x_k\rangle \colon R(x_1, \ldots .x_k)\}$.

It is sometimes convenient to abuse terminology and say that the predicate R has a property to mean that the language L_R that it represents has that property; in this way, we can say that the predicate R is decidable or recognizable or in Π_4, instead of saying that L_R is decidable or recognizable or in Π_4.

Observation 17 *Kleene's T-predicate is decidable.*

This is because a Turing machine, given input $\langle M, x, c\rangle$, can decide whether $T(M, x, c)$ holds: it uses the universal Turing machine to simulate M on input x for as many steps as c encodes, checks that c is an accepting computation of M on x, and accepts if this is the case and rejects if not.

We note the following technicality. Let Σ be the alphabet used to encode elements of a set, say Turing machines or k-tuples of strings. The encoding rules are typically such that some strings in Σ^* are syntactically invalid encodings. It is useful to insist that *all* strings of Σ^*, even syntactically invalid ones, actually represent some element of the set. We do this by thinking of all syntactically invalid strings as representing some default element of the set, typically a "simple" one. For example, in our encodings of Turing machines we thought of syntactically invalid strings as representing the Turing machine that rejects all inputs in zero steps (i.e., its initial state is the same as the reject state). In this way, if L_R is the language that represents predicate R, $\overline{L_R}$ is the language that represents the predicate $\neg R$. Note that in this manner, although every object has an encoding, the default object chosen to be represented by all invalid strings has multiple encodings.

2.2 Characterization of level 1

Consider the following languages:

- $\textsc{Univ} = \{\langle M, x\rangle \colon M \text{ accepts } x\}$.
- $\overline{\textsc{Empty}} = \{\langle M\rangle \colon \mathcal{L}(M) \neq \varnothing\}$.
- $\textsc{Common} = \{\langle M_1, M_2\rangle \colon \mathcal{L}(M_1) \cap \mathcal{L}(M_2) \neq \varnothing\}$.

We have proved that the first two are recognizable (but undecidable), and it is easy to show that so is the third. Now consider the following ways of representing these languages with predicates expressed as formulas in first-order logic:

- UNIV is represented by $\exists c\ T(M, x, c)$, since M accepts x if and only if there is a an accepting computation c of M on x.
- $\overline{\text{EMPTY}}$ is represented by $\exists x \exists c\ T(M, x, c)$, since the language recognized by M is nonempty if there is some string x and an accepting computation c of M on x.
- COMMON is represented by $\exists x \exists c_1 \exists c_2\ \big(T(M_1, x, c_1) \wedge T(M_2, x, c_2)\big)$, since the languages of M_1 and M_2 have a nonempty intersection if and only if there is some string x and accepting computations c_1 and c_2 of M_1 and M_2 on x, respectively.

Notice the following about these three examples:

- Each language is represented by a formula that consists of one or more existential quantifiers applied to a predicate.
- The predicate that follows the existential quantifiers is decidable. This is immediate in the first two examples since T is decidable, as we have already pointed out. In the third example the quantifier-free part of the formula is the conjunction of two decidable predicates, which is therefore also decidable (recall that the intersection of decidable languages is decidable).
- The objects that belong (or not) to each language are encodings of the free variables of the corresponding formula. For instance, in the third example, the free variables of the formula are M_1 and M_2 (because the formula defines a predicate that depends on M_1 and M_2); and the elements of the language COMMON that the predicate represents are encodings of the pair of M_1 and M_2.

This is not an accident about the above three languages. As the following theorem states, it is true of all recognizable languages and only of recognizable languages!

Theorem 18 *A language L is recognizable (i.e., $L \in \Sigma_1$) if and only if $L = \{x\colon \exists y\ P(x, y)\}$ for some decidable predicate $P(x, y)$.*

The existentially quantified variable y in the above formula is sometimes called a ***certificate***. This is because, through the predicate P, which is called a ***verifier***, it confirms x's membership in the language L: if a certificate y exists so that $P(x, y)$ is satisfied, then x is in L; and, conversely, if no such y exists, then x is not in L.

PROOF.
IF: Suppose L is a language such that $L = \{x\colon \exists y P(x, y)\}$, for a decidable predicate P. Let M_P be a Turing machine that decides P. Then the following TM is a recognizer for L:

```
   L-RECOGNIZER := on input x:
1      for each y in shortlex order do
2          run M_P on ⟨x, y⟩
3          if M_P accepts then accept
```

If $x \in L$ then, by definition of L, there is some y such that $P(x, y)$ holds. Therefore the decider M_P on input $\langle x, y\rangle$ accepts, and so L-RECOGNIZER accepts x. If $x \notin L$ then there is no y such that $P(x, y)$ holds and so L-RECOGNIZER on input x will loop and it does not accept x. Therefore L-RECOGNIZER is indeed a recognizer for L.

ONLY IF: Suppose L is a recognizable language. Thus, there is a Turing machine, say M_L, that recognizes L. We want to find a decidable predicate $P(x, y)$ such that $x \in L$ if and only if $\exists y\ P(x, y)$. What would constitute a certificate y that confirms x's membership in L? But, of course, an accepting computation of M_L on x would! So, we have that $x \in L$ if and only if $\exists c\ T(M_L, x, c)$, and by Observation 17 we have a formula of exactly the desired form. (Note that $T(M_L, x, c)$ is actually a predicate with only two variables, x and c, since the first argument is instantiated with the specific Turing machine M_L.) □

You may be wondering about the discrepancy between the statement of this theorem, which asserts that there is a formula with a *single* existential quantifier that represents a recognizable language, and the formulas for $\overline{\text{EMPTY}}$, which has two existentially quantified variables, and COMMON, which has three. There is in fact no discrepancy, since we can encode three variables x, y, z as a triple $\langle x, y, z\rangle$, and refer to x, y, and z by decoding $\langle x, y, z\rangle$. So, we could write the formula for COMMON as

$$\exists\langle x, c_1, c_2\rangle\ \big(T(M_1, x, c_1) \wedge T(M_2, x, c_2)\big). \tag{2}$$

More accurately, we could write it as

$$\exists y\ T'(M_1, M_2, y)$$

where $T'(M_1, M_2, y)$ is the predicate that is true if and only if y decodes into the three elements x, c_1, c_2, where c_1 is an accepting computation of M_1 on x and c_2 is an accepting computation of M_2 on x. The predicate $T'(M_1, M_2, y)$ is decidable because a Turing machine can decode the string y into the unique three strings x, c_1, and c_2, that y encodes and then use the universal Turing machine to check that c_1 and c_2 are indeed accepting computations of M_1 and M_2, respectively, on x. Henceforth we will write formulas such as (2) to indicate the combination of multiple variables quantified by the same quantifier — all $\exists$ or all $\forall$ — into a single variable under that quantifier.

Theorem 18 is very useful as it allows us to quickly classify certain languages as recognizable: All we have to do is find a (correct!) description of the language by a formula that has the proper form: an existentially quantified formula of a decidable predicate. For example, consider the language

$$B = \{\langle M\rangle\text{: } M \text{ accepts a string whose length is exactly equal to the number of states of } M\}.$$

This language is recognizable because

$$B = \Big\{\langle M\rangle\text{: } \exists x \exists c\, \big(\underbrace{T(M, x, c)}_{(1)} \wedge \underbrace{|x| = \text{number of states of } M}_{(2)}\big)\Big\}$$

Note that the predicates (1) and (2) are both decidable ((1) by Observation 17, and a Turing machine can certainly decide (2) given $\langle M, x, c\rangle$) and therefore so is their conjunction. Thus B can be expressed as an existentially quantified formula of a decidable predicate (the two existentially quantified variables can be combined into one, as we have seen) and so by Theorem 18, B is recognizable. *Note that B is undecidable, but Theorem 18 does not allow us to conclude this.* All that theorem says is that B is recognizable; some recognizable languages are decidable and some, like B, are not. (You can prove that B is undecidable using the Recursion Theorem.)

Co-recognizable languages

By definition, language L is co-recognizable if and only if $\overline{L}$ is recognizable. By Theorem 18 L is co-recognizable if and only if there is a decidable predicate P such that $x \in L \Leftrightarrow \neg\exists y\ P(x, y)$. But, $\neg\exists y\ P(x, y)$ is logically equivalent to $\forall y\ \neg P(x, y)$. Furthermore, $P(x, y)$ is decidable if and only if $\neg P(x, y)$ is decidable (simply by swapping the accept and reject states of any Turing machine that decides $P(x, y)$). Therefore we have the following characterization for co-recognizable languages:

Corollary 19 *A language L is co-recognizable (i.e., $L \in \Pi_1$) if and only if there is a decidable predicate $P(x, y)$ such that $L = \{x : \forall y\ P(x, y)\}$.*

Consider, for example, the language EMPTY of codes of Turing machines that accept no input: EMPTY $= \{\langle M\rangle : \mathcal{L}(M) = \varnothing\}$. We have proved that this is a co-recognizable language (by giving a deterministic Turing machine that recognizes its complement $\overline{\text{EMPTY}}$ using dovetailing or by giving a nondeterministic Turing machine that recognizes $\overline{\text{EMPTY}}$ by "guessing" a string and verifying that it is accepted by M) but here is another proof of this fact, based on Corollary 19, simply by noting that EMPTY $= \{\langle M\rangle : \forall x \forall c\ \neg T(M, x, c)\}$. Since $T(M, x, c)$ is decidable so is $\neg T(M, x, c)$ and we can combine the two universally quantified variables into one, as usual. So, by Corollary 19, EMPTY is a co-recognizable language.

2.3 Characterization of higher levels of the hierarchy

It is nice and useful to have this logical characterizable of the languages in the first level of the hierarchy. As we will see next, each level of the hierarchy is characterized exactly by logical formulas adhering to a specific format of increasing complexity as the level increases!

Lemma 20 *The language* FIN $= \{\langle M\rangle : \mathcal{L}(M)$ *is finite*$\}$ *is in* Σ_2.

PROOF. Consider the language

$$A = \{\langle M, x\rangle : M \text{ accepts some input } y \text{ that is after } x \text{ in shortlex order}\}.$$

It is easy to see that A is recognizable. (We can prove this by giving a Turing machine that recognizes it or by noting that A is represented by the following formula

$$\exists \langle y, c\rangle \underbrace{\big(y \text{ is after } x \text{ in shortlex order } \wedge\ T(M, y, c)\big)}_{\text{decidable predicate of } M, x, y, c}$$

so, since the quantifier-free part of the formula is a decidable predicate (of M, y, z, and c), Theorem 18 immediately implies that $A \in \Sigma_1$.)

Consider now the following OTM that uses A as an oracle.

```
1  on input ⟨M⟩:
2      for each x in shortlex order do
3          write ⟨M, x⟩ on the query tape
4          enter the query state
5          if the response of the oracle is "no" then accept
```

- If M accepts a finite set of strings, then there is some string x such that M accepts no string that is after x in shortlex order. When the loop is executed for the first such x, the oracle will respond "no" and the OTM accepts $\langle M\rangle$.
- If M accepts an infinite set of strings, then for every x there is some string y that is after x is shortlex order and M accepts y. So in that case the oracle answers "yes" in every iteration, the while loop never ends, and the OTM does not accept $\langle M\rangle$.

Therefore, the above OTM with oracle A recognizes FIN. Since $A \in \Sigma_1$, by Definition 1, FIN $\in \Sigma_2$. □

To say that the language of a Turing machine M is finite is to say that there is some string x so that M accepts no string y that comes after y in shortlex order. Thus FIN can be represented by the formula

$$\exists x \forall \langle y, c\rangle\ \underbrace{\big(y \text{ is after } x \text{ in shortlex order} \rightarrow \neg T(M, y, c)\big)}_{\text{decidable predicate of } M, x, y, c}$$

Note that the predicate in the quantifier-free part of the formula, under the brace, is decidable: Given $\langle M \rangle$, x, y, and c, a Turing machine can check whether y is after x in shortlex order and then, using the universal Turing machine, check whether c is an accepting computation of M on y.

The format of the formula that represents the Σ_2 language FIN is not an accident. It turns out that the pattern established by Theorem 18 for Σ_1 is extended for Σ_2 and, in fact, for all levels of the Arithmetic Hierarchy:

- A language L is in Σ_2 if and only if $L = \{x : \exists y_1 \forall y_2 \, P(x, y_1, y_2)\}$, for some decidable predicate P,
- a language L is in Σ_3 if and only if $L = \{x : \exists y_1 \forall y_2 \exists y_3 \, P(x, y_1, y_2, y_3)\}$, for some decidable predicate P,
- a language L is in Σ_4 if and only if $L = \{x : \exists y_1 \forall y_2 \exists y_3 \forall y_4 \, P(x, y_1, y_2, y_3, y_4)\}$, for some decidable predicate P,
- ...

  and, in general,
- a language L is in Σ_i if and only if $L = \{x : \exists y_1 \forall y_2 \exists y_3 \ldots \mathbf{Q} y_i \, P(x, y_1, y_2, y_3, \ldots y_i)\}$, and for some decidable predicate P, where the quantifier $\mathbf{Q}$ is $\forall$ if i is even, and $\exists$ if i is odd.

By complementing the language, negating the formula, and noting that if a predicate is decidable then so is its negation, we also have:

- A language L is in Π_i if and only if $L = \{x : \forall y_1 \exists y_2 \forall y_3 \ldots \mathbf{Q} y_i \, P(x, y_1, y_2, y_3, \ldots y_i)\}$, for some decidable predicate P, where the quantifier $\mathbf{Q}$ is $\forall$ if i is odd, and $\exists$ if i is even.

A few remarks about are in order before we prove this important and useful fact.

(1) The logical formulas that describe the predicates are in ***prenex normal form***, i.e., they consist of a sequence of quantifiers, followed by a quantifier-free formula. You know from CSCB36 that every predicate logic formula is equivalent to a formula in prenex normal form.

(2) The quantifier-free part of the formula defines a predicate that is decidable: There is a Turing machine that, given specific values for all the arguments, can determine whether the predicate is true or not for those values.

(3) For languages in Σ_i, the string of quantifiers starts with $\exists$ and then the type of quantifiers alternate between $\exists$ and $\forall$; for languages in Π_i, the string of quantifiers starts with $\forall$ and then the type of quantifiers alternates.

(4) What determines the level i is the number of alternations, not the number of quantifiers. This is because, as we mentioned earlier, we can encode a run of similarly-quantified variables by a single quantified variable; for example, $\forall y_{i1} \forall y_{i2} \ldots \forall y_{ik}$ can be written as $\forall y_i$, where $y_i = \langle y_{i1}, y_{i2}, \ldots, y_{ik} \rangle$; and similarly for a run of existential quantifiers.

Notation 21 *For every positive integer i we will use the symbol $\boldsymbol{Q}_i$ to stand for the quantifier $\exists$ if i is odd and for the quantifier $\forall$ if i is even; the symbol $\overline{\boldsymbol{Q}}_i$ stands for the dual quantifier: $\forall$ if i is odd and $\exists$ if i is even.*

Theorem 22 (Kleene) *For every $i \geq 1$, a language L is in*

(a) Σ_i *if and only if* $L = \{x : \exists y_1 \forall y_2 \exists y_3 \ldots \boldsymbol{Q}_i y_i \; P(x, y_1, y_2, y_3, \ldots y_i)\}$, *and*

(b) Π_i *if and only if* $L = \{x : \forall y_1 \exists y_2 \forall y_3 \ldots \overline{\boldsymbol{Q}}_i y_i \; P(x, y_1, y_2, y_3, \ldots y_i)\}$,

where P is some decidable predicate.

The following lemma generalizes Theorem 18 and does most of the work for the proof of Theorem 22.

Lemma 23 *For every $i \geq 1$, a language L is in Σ_i if and only if $L = \{x : \exists y P(x, y)\}$, where P is a Π_{i-1} predicate.*[5]

Proof of Lemma 23. Use induction on i, generalizing ideas in the proof of Theorem 18.

The basis, $i = 1$, is just Theorem 18, since a Π_0 predicate is, by definition of Π_0, decidable. For the induction step, let $i \geq 2$ and suppose that the lemma holds for $i - 1$; we will prove that it holds for i.

If: Suppose $L = \{x : \exists y \; P(x, y)\}$ where $P(x, y)$ is a Π_{i-1} predicate. We will prove that $L \in \Sigma_i$ by constructing an OTM that accepts L using an oracle in Σ_{i-1}.

Recall that to say $P(x, y)$ is a Π_{i-1} predicate means that the language $L_P = \{\langle x, y\rangle : P(x, y)\}$ is in Π_{i-1}. Thus, $\overline{L_P} = \{\langle x, y\rangle : P(x, y)\}$ is in Σ_{i-1}. Consider the following OTM, using oracle $\overline{L_P}$, where y_i is the ith string in shortlex order:

```
1  on input x:
2      for each i := 1, 2, 3, ... do
3          write ⟨x, y_i⟩ on the query tape
4          enter the query state
5          if the response of the oracle is "no" then accept
```

We now show that this OTM using oracle $\overline{L_P}$ recognizes L.

- If $x \in L$, by definition of L, there is some string y such that $P(x, y)$. Therefore, in some iteration of the while loop the string $\langle x, y\rangle$ written on the query tape belongs to L_P, and thus the oracle for $\overline{L_P}$ will reply "no", and the OTM accepts x.
- If $x \notin L$, by definition of L, there is no string y such that $P(x, y)$. Therefore, in every iteration of the while loop the string $\langle x, y\rangle$ written on the query tape does not belong to L_P, and so the oracle for $\overline{L_P}$ will reply "yes" and the loop will not terminate. Thus, the OTM does not accept x.

This shows that the above OTM with oracle $\overline{L_P} \in \Sigma_{i-1}$ recognizes L, so $L \in \Sigma_i$, as wanted.

Only If: Now suppose that $L \in \Sigma_i$. We want to show that there is a predicate P in Π_{i-1} such that

$$x \in L \quad \Leftrightarrow \quad \exists y \; P(x, y). \tag{3}$$

Since $L \in \Sigma_i$ there is an OTM M that recognizes L using an oracle $A \in \Sigma_{i-1}$. By the induction hypothesis applied to language A, there is a Π_{i-2} predicate R such that

$$y \in A \quad \Leftrightarrow \quad \exists v \; R(y, v), \tag{4}$$

$$\text{and therefore} \quad z \notin A \quad \Leftrightarrow \quad \forall w \; \neg R(z, w). \tag{5}$$

Roughly speaking, the predicate $P(x, y)$ we are looking for, that satisfies (3), asserts that y encodes an accepting computation c of M on x using A.

To see how to do this, let us consider such an accepting computation c of M on x using A. It consists of a finite sequence of configurations. A ***step*** of the computation is the transition between successive

[5] Recall that "P is a Π_{i-1} predicate" is an abbreviation for "the language L_P represented by the predicate P is in Π_{i-1}".

configurations in this sequence. We call a step from configuration C to configuration C' ***regular***, if the OTM is not in the query state $q_?$ in configuration C; we call it a ***yes query step*** (respectively, ***no query step***), if the OTM is in the query state in C and responds "yes" (respectively "no") in C'. To be an accepting computation of M on x using oracle A, c must satisfy the following conditions:

(a) It starts with the initial configuration of M on x;

(b) it ends with an accepting configuration of M;

(c) every *regular* step of c is consistent with the transition function of M;

(d) in every yes query step of c, the string in the query tape before the step belongs to A; and

(e) in every no query step of c, the string in the query tape before the step does not belong to A.

As we noted when we justified Observation 17, (a)-(c) can be checked by a Turing machine; (d) and (e), however, cannot necessarily be checked by a Turing machine because A is a language in Σ_{i-1} and is therefore quite possibly undecidable. To check (d) and (e) we will use the predicate R that acts as a verifier for A, and leverage (4) and (5) to construct the predicate P that acts as a verifier for L.

Consider the predicate $S(M, x, c, v, w)$ defined to be true if and only if all of the following hold:

(1) c encodes a finite sequence of configurations of M that starts with the initial configuration of M on x, ends with an accepting configuration of M, and each configuration other than the first is obtained from the previous configuration by a valid move of M.

(2) c has some number k of yes query steps with strings $y_1, y_2, \ldots, y_k$ on the query tape, and some number ℓ of no query steps with strings $z_1, z_2, \ldots, z_\ell$ on the query tape.

(3) v encodes a sequence of k strings $v_1, v_2, \ldots, v_k$.

(4) w encodes a sequence of ℓ strings $w_1, w_2, \ldots, w_\ell$.

(5) For each $i \in [1..k]$, $R(y_i, v_i)$ holds; i.e., v_i certifies that $y_i \in A$.

(6) For each $i \in [1..\ell]$, $\neg R(z_i, w_i)$ holds; i.e., w_i does *not certify* that $z_i \in A$.[6]

Since the predicate R is in Π_{i-2}, the language it represents $L_R = \{\langle y, v\rangle\colon R(y, v)\}$ is also in Π_{i-2}; therefore $\overline{L_R}$ is in Σ_{i-2}. There is an OTM N that *decides* (not merely recognizes) $S(M, x, c, v, w)$, using the Σ_{i-2} oracle $\overline{L_R}$: For (1)-(4) N does not need an oracle, and for (5) and (6) it uses $\overline{L_R}$: To determine whether $R(y_i, v_i)$ is true, N extracts y_i from the sequence of configurations that c encodes, writes the pair $\langle y_i, v_i\rangle$ on the query tape, and enters the query state; the $\overline{L_R}$ oracle responds "no" if and only if $R(y_i, v_i)$ is true, and so N can determine whether (5) holds. Similarly, to determine whether $R(z_i, w_i)$ is true, N extracts z_i from c, writes the pair $\langle z_i, w_i\rangle$ on the query tape, and enters the query state; the $\overline{L_R}$ oracle responds "yes" if and only if $\neg R(z_i, v_i)$ is true, and so N can determine whether (6) holds. Thus, the language L_S represented by S is decided by the OTM N using $\overline{L_R}$, and therefore the language $\overline{L_S}$ represented by $\neg S$ is decided by an OTM using $\overline{L_R}$ (merely by swapping the accept and reject states of N). Therefore,

$$\text{the predicate } \neg S(M, x, c, v, w) \text{ is decidable by an OTM using oracle } \overline{L_R}. \tag{6}$$

Claim 24 *Let $P'(x, y)$ be any predicate that is decidable by an OTM using some oracle B. Then the predicate $\exists y\ P'(x, y)$ is recognizable by an OTM using B.*

[6] Be sure to understand the difference between w_i *not certifying* that z_i is a member of A (which is what we claim here), and w_i certifying that z_i is *not* a member of A.

PROOF OF CLAIM 24. We define a recognizer that uses B as follows: It runs through all possible y and checks whether $P'(x,y)$ holds using the OTM decider for $P'(x,y)$; if the decider accepts, then our recognizer accepts x, as it is true that the predicate $\exists y\, P'(x,y)$ is true; if the decider rejects, we continue with the next y. So, if x satisfies the predicate $\exists y\, P'(x,y)$, then the OTM accepts x; and if x does not satisfy the predicate $\exists y\, P'(x,y)$, the recognizer will loop and does not accept x. □

Applying Claim 24 to (6), we conclude that $\exists w\ \neg S(M,x,c,v,w)$ is recognizable using oracle $\overline{L_R}$; and since $\overline{L_R}$ is in Σ_{i-2}, the predicate $\exists w\, \neg S(M,x,c,v,w)$ is in Σ_{i-1}. Thus, its negation is in Π_{i-1}:

$$\text{the predicate } \forall w\, S(M,x,c,v,w) \text{ is in } \Pi_{i-1}. \tag{7}$$

Finally, consider the predicate

$$\exists \langle c,v\rangle \forall w\, S(M,x,c,v,w).$$

The free variables of this formula are M and x, and the formula states that there is a computation c and a string v so that for all strings w the following are true:

(1) c encodes a finite sequence of configurations of M that starts with the initial configuration of M on x, ends with and accepting configuration of M, and each configuration other than the first is obtained from the previous configuration by a valid move of M.

(2) c has some number k of yes query steps with strings $y_1, y_2, \ldots, y_k$ on the query tape, and some number ℓ of no query steps with strings $z_1, z_2, \ldots, z_\ell$ on the query tape.

(3) v encodes a sequence of k strings $v_1, v_2, \ldots, v_k$.

(4) w encodes a sequence of ℓ strings $w_1, w_2, \ldots, w_\ell$.

(5) For each $i \in [1..k]$, $R(y_i, v_i)$ holds; i.e., v_i certifies that $y_i \in A$.

(6) For each $i \in [1..\ell]$, $\neg R(z_i, w_i)$ holds; i.e., w_i does *not certify* that $z_i \in A$.

In other words, the predicate is true if and only if there is an accepting computation of the OTM M on input x using oracle A, i.e., if and only if $x \in L$. So,

$$x \in L \quad \Leftrightarrow \quad \exists \langle c,v\rangle \underbrace{\forall w\, S(M,x,c,v,w)}_{\text{in } \Pi_{i-1} (see\ (7))}.$$

Thus, taking $P(x,\langle c,v\rangle)$ to be $\forall w\, S(M,x,c,v,w)$, we have that

$$x \in L \quad \Leftrightarrow \quad \exists \langle c,v\rangle\, P(x,\langle c,v\rangle)$$

for some Π_{i-1} predicate P, as wanted. This concludes the proof of Lemma 23. □

From Lemma 23 we immediately get

Corollary 25 *For every $i \geq 1$, a language L is in Π_i if and only if $L = \{x :\ \forall y\, P(x,y)\}$, where P is a Σ_{i-1} predicate.*

Finally, using Lemma 23 and Corollary 25 , we can give the

PROOF OF THEOREM 22. By induction on i. The base case $i = 1$ is just Theorem 18 and Corollary 19. For the induction step, consider any $i > 1$ and assume that the Theorem holds for $i-1$; we will prove that it holds for i.

For part (a), consider any language $L \in \Sigma_i$. By Lemma 23,

$$x \in L \;\Leftrightarrow\; \exists y_1 P''(x, y_1), \text{ for some } \Pi_{i-1} \text{ predicate } P''. \tag{8}$$

By definition of the language $L_{P''}$ represented by the predicate P'' and part (b) of the induction hypothesis applied to $L_{P''}$, we have

$$P''(x, y_1) \;\Leftrightarrow\; \langle x, y_1 \rangle \in L_{P''} \;\Leftrightarrow\; \forall y_2 \exists y_3 \ldots \mathbf{Q}_i y_i \; P'(\langle x, y_1 \rangle, y_2, y_3, \ldots, y_i) \tag{9}$$
$$\text{for some decidable predicate } P'.$$

Now define the predicate $P(x, y_1, y_2, y_3, \ldots, y_i)$ to be true if and only if $P'(\langle x, y_1 \rangle, y_2, y_3, \ldots, y_i)$ is true. (That is, P is an $(i+1)$-place predicate that has the same truth value as the i-place predicate P' when we replace in the first two arguments x and y_1 of P by their encoding as a pair $\langle x, y_1 \rangle$ and plug this, along with the remaining $i-1$ arguments of P, into P'.) Since P' is decidable, so is P.

From this, (8) and (9) we have

$$\begin{aligned} x \in L \;\Leftrightarrow\; \exists y_1 P''(x, y_1) &\;\Leftrightarrow\; \exists y_1 \forall y_2 \exists y_3 \ldots \mathbf{Q}_i y_i \; P'(\langle x, y_1 \rangle, y_2, y_3, \ldots, y_i) \\ &\;\Leftrightarrow\; \exists y_1 \forall y_2 \exists y_3 \ldots \mathbf{Q}_i y_i \; \underbrace{P(x, y_1, y_2, y_3, \ldots, y_i)}_{\text{decidable}} \end{aligned}$$

which proves part (a).

Part (b) follows by noting that $L \in \Pi_i$ if and only if $\overline{L} \in \Sigma_i$, representing $\overline{L}$ by a predicate according to part (a), and negating that predicate to get one that represents L. □

2.4 Placing some languages in the hierarchy

Theorem 22 is very helpful in placing languages in the arithmetic hierarchy. Given the definition of a language, we look to represent that language using a formula in prenex normal form whose quantifier-free part involves decidable predicates; the number i of quantifier alternations determines the level at which the formula places the language and the type of the leading quantifier ($\exists$ versus $\forall$) determines whether the language is in Σ_i or Π_i. We strive for the simplest (correct!) formula, to place the language in the lowest (most restrictive) set. We have seen a few examples of this process already; we demonstrate it further with more examples.

Some languages in level 2 of the arithmetic hierarchy

We have seen that FIN is in Σ_2 based on two different approaches: First, we proved it by giving a recognizer that uses a Σ_1 oracle (see Lemma 20 and its proof). We also gave a Σ_2-type predicate that represents FIN, namely

$$\exists x \forall y \forall c \, \underbrace{\big(y \text{ is after } x \text{ in shortlex order} \rightarrow \neg T(M, y, c)\big)}_{\text{decidable predicate of } M, x, y, c}.$$

By definition of Π_2, the complement of FIN, INFIN, is in Π_2. We can also see this by noting that INFIN is represented by the following formula:

$$\forall x \exists y \exists c \, \underbrace{\big(y \text{ is after } x \text{ in shortlex order} \wedge T(M, y, c)\big)}_{\text{decidable predicate of } M, x, y, c}$$

which expresses the fact that M accepts an infinite number of string by asserting that for every string x, however large in shortlex order, there is an even larger y that M accepts.

As another example of a Π_2 language consider $\text{ALL} = \{\langle M\rangle\colon \mathcal{L}(M) = \Sigma_M^*\}$ (where Σ_M denotes the input alphabet of Turing machine M). We can express this language as the set of codes of Turing machines M so that for every string x there is an accepting computation c of M on input x. So, ALL is represented by the predicate $\forall x \exists c\ T(M, x, c)$. Since this has two alternations of quantifiers, starting with a universal one, the corresponding language is in Π_2. Since ALL is in Π_2 its complement $\overline{\text{ALL}}$ is in Σ_2. By Theorem 22 this can also be seen by the fact that $\overline{\text{ALL}}$ is represented by the predicate $\exists x \forall c\ \neg T(M, x, c)$: there is some string x such that every c is not an accepting computation of M on x.

Consider the language $\text{EQUIV} = \{\langle M_1, M_2\rangle\colon \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$. To say that two Turing machines recognize the same language is to say that for each string x either both TMs do not accept x or both accept x. So EQUIV is represented by the predicate

$$\forall x \forall c_1 \forall c_2 \exists c_3 \exists c_4 \underbrace{\Big(\big(\neg T(M_1, x, c_1) \wedge \neg T(M_2, x, c_2)\big) \vee \big(T(M_1, x, c_3) \wedge T(M_2, x, c_4)\big)\Big)}_{\text{decidable predicate of } M_1, M_2, x, c_1, c_2, c_3, c_4}$$

which, according to Theorem 22, places EQUIV in Π_2. EQUIV is also represented by the predicate

$$\forall x\, \underline{\exists c_3 \exists c_4}\, \underline{\forall c_1 \forall c_2} \underbrace{\Big(\big(\neg T(M_1, x, c_1) \wedge \neg T(M_2, x, c_2)\big) \vee \big(T(M_1, x, c_3) \wedge T(M_2, x, c_4)\big)\Big)}_{\text{decidable predicate of } M_1, M_2, x, c_1, c_2, c_3, c_4}$$

which differs from the previous one only in the order of the underlined quantifiers. This formulation puts EQUIV in Π_3!

There is no contradiction here: We know from Theorem 16 that $\Pi_2 \subseteq \Pi_3$. The second of these choices represents EQUIV with more quantifier alternations than is necessary. In general, the lowest possible level of the hierarchy, being more restrictive, gives us more information about the language. So, in general, we try to represent a language with the fewest possible number of quantifier alternations. We will return briefly to this phenomenon later (see subsection entitled "Complete languages").

It is important to understand that <u>in general, we cannot move quantifiers around willy-nilly</u> without changing the meaning of the formula. In the case of the two formulas that represent EQUIV we could switch the order of the quantifiers for c_1, c_2, c_3, c_4 because, roughly speaking, each of them refers only to a different part of the quantifier-free part of the formula. In general, however, $\forall x \exists y\, P(x, y)$ is <u>not</u> equivalent to $\exists y \forall x\, P(x, y)$. For example, if $P(x, y)$ means "x is before y in shortlex order", then $\forall x \exists y\, P(x, y)$ is true but $\exists y \forall x\, P(x, y)$ is false.

Placing a language in Δ_2

We have seen in Theorem 15(a) that $\Delta_i = \Sigma_i \cap \Pi_i$. Thus, one way of placing a language in Δ_i is to find two formulas that represent it, one placing it in Σ_i and one placing it in Π_i. Consider the language $\text{UNIQUE} = \{\langle M\rangle\colon |\mathcal{L}(M)| = 1\}$. This can be represented by the formula

$$\exists x \exists c_x \forall y \forall c_y \underbrace{\Big(T(M, x, c_x) \wedge \big(y \neq x \rightarrow \neg T(M, y, c_y)\big)\Big)}_{\text{decidable predicate of } M, x, c_x, y, c_y}$$

where of course we can combine the two existential quantifiers into one $\exists\langle x, c_x\rangle$ and the two universal ones into one $\forall\langle y, c_y\rangle$, and then have the encoded pairs be decoded in the (still decidable) quantifier-free part of the formula. This formula says that there is an accepting computation of M on some input x and that all <u>other</u> inputs y have no accepting computation by M. Since there are two quantifier alternations starting with $\exists$, this places UNIQUE into Σ_2.

UNIQUE can also be represented by the formula

$$\forall x \forall c_x \forall y \forall c_y \exists z \exists c_z \underbrace{\Big(\big(T(M, x, c_x) \wedge T(M, y, x_y) \rightarrow x = y\big) \wedge T(M, z, c_z)\Big)}_{\text{decidable predicate of } M, x, c_x, y, c_y, x, c_z}.$$

This expresses the fact that M accepts exactly one string in a different way: Any two strings x and y that M accepts are the same, and there is some string z that M accepts. The form of this predicate places UNIQUE into Π_2, since there are two quantifier alternations and the leading quantifier is $\forall$. Since UNIQUE is in both Σ_2 and Π_2, it is in Δ_2.

Some languages in level 3 of the arithmetic hierarchy

A language is called ***cofinite*** if its complement is finite. Clearly every cofinite language is infinite, but not every infinite language is cofinite. For example, the set of even-length strings over some alphabet is an infinite language that is not cofinite since its complement, the set of odd-length strings, is also infinite. A language is cofinite if, starting with some string x, it contains every string y after x in shortlex order. So, the set COFIN $= \{\langle M\rangle$: $\mathcal{L}(M)$ is cofinite$\}$ is represented by the formula

$$\exists x \forall y \exists c \underbrace{\big(y \text{ is after } x \text{ in shortlex order} \rightarrow T(M,y,c)\big)}_{\text{decidable predicate of } M,x,y,c}.$$

This places COFIN into Σ_3.

Consider the language REG $= \{\langle M\rangle$: $\mathcal{L}(M)$ is regular$\}$, i.e., the set of codes of Turing machines that recognize regular languages. Just as we can encode Turing machine as strings, we can encode finite state automata (FSA) as strings by establishing conventions for how to represent states, symbols, and the transition function, and how to identify the start state and the accepting states of the automaton. Without giving the details of such an encoding, let us assume an arbitrary, reasonable one; by "reasonable" we mean an encoding so that, given the code $\langle A\rangle$ of a FSA A, a Turing machine can determine the component parts of the FSA (e.g., it can determine the code of the initial state) and can use the encoding of the transition function to simulate the operation of A on any input string x.[7] We can then represent the language REG by the following formula:

$$\exists \langle A\rangle \forall x \exists c \underbrace{\big(A \text{ is a FSA that accepts } x \leftrightarrow T(M,x,c)\big)}_{\text{decidable predicate of } M,A,s,c}.$$

This states that there is a FSA A that accepts the same strings as M, i.e., M recognizes the same language as some FSA. Since a language is regular if and only if it is accepted by a FSA, the above formula represents REG, and this places REG in Σ_3: the formula has three quantifier alternations starting with an existential quantifier applied to a decidable quantifier-free predicate. (Recall that by our assumption of a reasonable encoding for FSA, given $\langle A\rangle$ and x, a Turing machine can determine if A accepts x or not; there is no danger of looping in FSA!)

Consider the language UNDEC $= \{\langle M\rangle$: $\mathcal{L}(M)$ is undecidable$\}$, i.e., the set of codes of Turing machines that recognize undecidable languages. A language is undecidable if every Turing machine that recognizes it loops on some input. Let $H(M,x,c)$ be the predicate that is true if and only if c is a halting computation of Turing machine M on input x; i.e., c takes M from the initial configuration q_0x to a configuration where M's state is the accept or reject state. Like Kleene's T-predicate, $H(M,x,c)$ is decidable. So, we can represent UNDEC by the following predicate

$$\forall \langle M'\rangle \forall x \forall c_1 \forall c_2 \exists c_3 \exists c_4 \exists y \forall c_5 \Bigg(\underbrace{\Big(\big(\neg T(M,x,c_1) \wedge \neg T(M',x,c_2)\big) \vee \big(T(M,x,c_3) \wedge T(M',x,c_4)\big)\Big)}_{M \text{ and } M' \text{ are equivalent: for every } x, \text{ both don't accept it or both accept it}} \rightarrow \neg H(M',y,c_5)\Bigg).$$

[7] An example of an *un*reasonable encoding would be one that includes rules such as "a state of the FSA is an accepting state if and only if it is encoded by a string that also represents a Turing machine that halts on empty tape". These are logically sound rules, but a Turing machine cannot use them to simulate the operation of the FSA.

It states that every Turing machine M' that is equivalent to M (every string is either rejected by both or accepted by both) fails to halt on some input y — i.e., M' is not a decider. This places UNDEC in Π_3.

Complete languages

Recall our example of EQUIV, which one formula placed in Π_2 and another in Π_3. As we explained, the first formula gives us more accurate information about the actual "degree of unsolvability" of this language than the second. But this raises the question: can we know when the predicate that represents a language gives us the most accurate information? In terms of Theorem 22 this means expressing it with the fewest quantifier alternations. How do we know, for example, that there is no way of representing EQUIV with just one kind of quantifier (all existential or all universal), placing it in level 1?

One way of doing this is by using the concept of complete languages, which we introduced earlier to prove that every level of the arithmetic hierarchy contains new languages, inherently "more undecidable" than those in lower levels. For example, if we show that EQUIV is a complete language for Π_2, which as you will recall intuitively means that EQUIV is one of the "hardest" languages in Π_2, then EQUIV cannot also be in Π_1.

To see why this is the case, we first note the following:

Theorem 26 *Let L, L' be languages such that $L \leq_m L'$, and A be any language.*

(a) If L' is recognizable using oracle A then so is L; equivalently, if L is not recognizable using A then neither is L'.

(b) If L' is decidadable using oracle A then so is L; equivalently, if L is not decidadable using A then neither is L'.

This generalizes facts we have already seen in relation to regular (non-oracle) Turing machines, and is proved in a very similar way (left as an exercise to remind you the proofs of these earlier results).

Theorem 27 *For all $i > j \geq 1$, if L is Σ_i-complete (respectively, Π_i-complete) then $L \notin \Sigma_j$ (respectively, Π_j).*

PROOF. We will show the proof for Σ; the proof for Π is similar.

Suppose, for contradiction, that L is a Σ_i-complete language, yet $L \in \Sigma_j$ for some $j < i$. Since L is Σ_i-complete and $H^i \in \Sigma_i$, we have $H^i \leq_m L$. Since $L \in \Sigma_j$ and H^j is Σ_j-complete, we also have $L \leq_m \Sigma^j$. So, by transitivity of $\leq_m$, $H^i \leq_m H^j$. Since $H^j \in \Sigma_j$, H^j is recognized by an OTM using some oracle in Σ_{j-1}; and since $H^i \leq_m H^j$, by Theorem 26(a), H^i is recognized by an OTM using some oracle in Σ_{j-1}, so $H^i \in \Sigma_j$, contradicting Theorem 15 (recall that H^i is in Σ_i and not in lower levels). □

As an application, consider the language FIN. We have already seen that this language is in Σ_2 — in fact we have proved this in two different ways, (a) by showing a recognizer for FIN that uses a Σ_1 oracle (see proof of Theorem 20) and (b) by representing the language with a formula of the appropriate form based on Theorem 22. We will prove that FIN is Σ_2-complete; so, by Theorem 27, level 2 is the lowest to which FIN belongs: it cannot be represented by a formula that involves fewer than two quantifier alternations.

Theorem 28 FIN *is Σ_2-complete.*

PROOF. By Theorem 20, FIN $\in \Sigma_2$. It remains to prove that for each $L \in \Sigma_2$, $L \leq_m$ FIN. Since $L \in \Sigma_2$, there is a decidable predicate $P(x, y, z)$ such that

$$x \in L \quad \Leftrightarrow \quad \exists y \forall z P(x, y.z). \tag{10}$$

Let M_P be a Turing machine that decides P. We will show a computable function f: $x \mapsto \langle M \rangle$ so that

(a) if $x \in L$ then $\mathcal{L}(M)$ is finite, and

(b) if $x \notin L$ then $\mathcal{L}(M)$ is infinite.

Given x, the computable function f produces the code $\langle M \rangle$ of the following Turing machine:

```
1  M := on input w:
2      for each y that precedes w in shortlex order do
3          for each z in shortlex order do
4              run M_P on ⟨x, y, z⟩
5              If M_P rejects (i.e., P(x, y, z) is false) then break    # continue with next y
6      accept
```

- If $x \in L$, by (10), there exists some y such that for all z, $P(x, y, z)$ is true. Let y_0 be the first such y in shortlex order. Consider the computation of M on any input w that is after y_0 in lexicographic order. By the choice of y_0, for every y preceding y_0 is lexicographic order there is some z such that $P(x, y, z)$ is false, so all iterations of the outer for-loop for y preceding y_0 will terminate by the conditional in line 5. When the outer for-loop is executed for y_0, the inner for-loop will not terminate, because M_P will accept input $\langle x, y_0, z \rangle$, for every z (since $P(x, y_0, z)$ is true for all z). Therefore M does not accept any string w that is after y_0 in shortlex order, and so $\mathcal{L}(M)$ is finite, as wanted for (a).

- If $x \notin L$, by (10), for every y there is some x such that $P(x, y, z)$ is false. Consider the computation of M on any input w. For each y considered in the outer for-loop, eventually the inner for-loop will be executed for some z such that $P(x, y, z)$ is false, and so the inner for-loop will terminate by the conditional in line 5. So, after being executed for all y that precede w in shortlex order, the outer for-loop will also terminate and M accepts w in line 6. Therefore, in this case M accepts every input w and so $\mathcal{L}(M)$ is infinite, as wanted for (b). □

Since FIN is complete for Σ_2, it is straightforward to argue that its complement, INFIN, is likewise complete for Π_2. It turns out that ALL is complete for Π_2 and therefore $\overline{\text{ALL}}$ is complete for Σ_2, and that COFIN, and REG are complete for Σ_3 (and so their complements complete for Π_3), and that UNDEC is complete for Π_3 (and so its complement DEC is complete for Σ_3). The proofs of these results are beyond the scope of this document.