Everything You Need To Know About MAT309

Nathanael Chwojko-Srawley

December 19, 2022

Contents

Abstract

In these notes, we will cover the language of first-order and predicate logic, define a new mathematical object known as a *model* and use it to define notions of truth, satisfiability, implication, and so forth. Then, we shall take a moment to detour from looking at "truth" and instead look at "deducibility" or "provability". This will culminates in linking "truth" and "provable" via the soundness and completeness theorems. After having covered the notion of provability, we shall start looking into computability (which, if you think about it, is the natural way in which we think), and show that something being "computable" is equivalent to it being "recursive", thus giving us a model for working with computable processes. Finally, we build up towards showing our two mains theorems: Gödels Incompleteness Theorem I and II. Essentially, if we have a system of axioms strong enough to be used to prove and work with the natural numbers, then there is no way of finding and proving all possible theorems, and there is no way of showing the system of axioms has a contradiction; these both are the foundation of the crisis in mathematics spearheaded by Gödel.

Some Philosophical Musings

Fundamentally, in any reality you might imagine (social, mental, physical), there is always an ontology and an epistemology, and no matter what we give symbols and sounds meaning through language to represent these, arguably even embody these. In math, the objects are usually called the *structures* or the *models*. We want to study as close as possible what is the "first principle" of objects. This naturally leads to many classical philosophical problems: what's the relation between epistemology and ontology, is there a first principle, what's the relation between logos and language, where do we derive meaning, is it possible to make analytic a-priori claims, and so forth.

Very often the same underlying object has many different interpretations depending on it's purpose, and it is suited for some purposes over other purposes. For example, let's take $\mathbb{R}^n$. Then

1. It is a vector-space: given an understanding of what a function is, we have that $+$ and $\mathbb{R} \cdot \mathbb{R}^n$ is well-defined. In the right number of dimensions, it is even an algebra (usually in $n = 1, 2, 4, 8$ we can impose an interesting multiplication structure)
2. It is a topological space: Given an understanding of subsets and closure, we have topology on it. There are many topologies that can even be put on it (from the obvious three, to many other possible one using weak topologies, order topologies, an so forth)
3. It is a metric space. There is a notion of distance. We can go further and say it's a normed and an inner product space, telling us that there is a notion of size and "orientation"
4. It is a complete space: we may freely use sequences without the worry of their limits not existing. Though it's not important, the space also complete through in terms of nets.
5. If is a smooth manifold: It is (trivially) locally Euclidean. The notion of differentiability is well-defined.
6. It is a flat space: The gaussian curvature is always 0, which also tells us that $\mathbb{R}^n$ does not exhibit the geometric properties of hyperbolic space or positively curved space.
7. It is classical Euclidean space (by eliminating the notion of an origin)
8. it is *not* a discrete space: results requiring the lack of continuity might not work on it
9. it is *not* a complex manifold of n is odd.
10. it is an ordered space (in the classical ordering) if $n = 1$

Thus, it is important to understand what epistemological structure we are considering on the object. As in the examples above, that often involves literally upgrading the object to have that extra information (ex.a vector-space, a topological space), however this is in my mind because we are trying to make all objects part of the universal epistemology of set theory, and so we would distinguish objects by adding structures to them. In a more "early mathematics" or pre-set theory of looking at it, we can think of $\mathbb{R}^n$ having all these properties (or not having for the false cases), and using the right epistemology to distinguish them.

Overall, we see how important it is to look at the epistemology. Importantly, we need a clear way of communicating and expressing this epistemology, i.e. a *language*. We usually will say *formal*

language, to mean an invented set of symbols to which we prescribe some meaning. We will take some time in the first chapter to describe the formal language we will use in these notes

(symbolic logic is when you block up concepts and look at the relations between them; Aristotle did this and tried to get rid of some Ethos and Pathos from reasoning)

1

Languages

Throughout history, the study of reason (and epistemology more generally, so source of meaning, mode of reasoning, nature of truth etc.) took on many forms. We will say that what we are studying when studying reason and all that surrounds reason is that we are studying *logos*. Though all of the following were happening at the same time, we can roughly say that there is a chronology in what philosophers believed to be the focus for the study of logos is:

1. Ontological focus: This started with the works of Thales (if not in a yet-to-be-found work in South Asia) who believed that "natural phenomena have natural causes", moving away from mythos based arguments and onto logos based arguments. These philosophers of the western tradition (essentially up to Descartes) focused on finding the meaning, the truth, the nature of valid reasoning (i.e. the logos) in the "real" world (what I would call the physical world, or physical cosmology). The word *ontology* comes from *ontos* meaning "that which is", "that has happened", or "that is true", and *logos* meaning "account of", or "understanding of". Key here is that language was used to express *opinions* of the world.

2. epistemological focus: We can say that this period starting with Descartes and continuing on to Frege was the start of the questioning of the validity of knowledge-claims. We may ask questions like "how do we know" or "how are we sure this is knowledge". This leads to philosophical movements like *rationalism*, and more interestingly and sophisticatedly *idealism*. Some examples of the consequence of this way of thinking is to have *methods* (ex. the scientific method) be used as the authority of knowledge, or to believe that the source of knowledge. While this project was a good project, the problem was that it is actually quite difficult to find a model for which to analyze such a system of knowledge. It turned out that the emphasis had to be on language, that is language naturally inherits.

3. Linguistic Focus: Starting with Frege's PhD thesis in the 1870s, it has become evident that language is deeply tied to any metaphysic – one simply doesn't have a non-metaphysical language. Hence, much more care must be taken in the words we use and pin down: their exact

meaning, the exact way to put them together, the exact scope of them, the exact rule to on how to use them, and so forth. The professor of this course also points out how logos no longer has any grounding in an authority, be it divine or other. There is also now a correlation with the loss of authority in Europe in general at this time, with the *crisis of meaning* and *nihilism* movements spreading.

1.1 First Order Formal Languages

The language we shall create will be made to ask about the validity or "truth" of structures that we may put on sets. For example, for groups we will want that $a * (b * c) = (a * b) * c$ for all elements, or for an ordered field we will want one of $x > 0$ or $x < 0$ or $x = 0$, but not any two at once, or that there is a special constant symbol 1 such that $1 * c = c$. First order language is designed to talk about such statements. The key features of the first order language will be:

1. constant: symbols that represent some value
2. relation: something that relates two symbols (the verbs if you will)
3. functions: something that can combine symbols together (the nouns or compound nouns if you will)
4. variables: symbols that can be substituted with some elements of your "set"

We have yet to bring sets in our discussion: this will happen once we will find *models* on that will let us use our language.

Definition 1.1.1: First-Order Language

The set $\mathcal{L}$ is an infinite collection of distinct symbols, no one of which is properly contained in another, separated into the following categories:

1. Parenthesis: which are the symbols $(,)$
2. Connectives: which are the symbols $\vee, \neg$ (possibly also $\wedge$). In some textbooks, the connectives are $\neg$ and $\rightarrow$.
3. Quantifier: which is the symbol $\forall$ (possibly also $\exists$)
4. Equality symbol: $=^a$
5. **Variables**, one for each positive integer n: $v_1, v_2, ..., v_n, ...$. The set of variable symbols will be denoted *Vars*
6. **Constant symbols**: Some set of zero or more symbols
7. **Function symbols**: For each positive integer n, some set of zero or more n-ary *function symbols*
8. **Relation symbols**: For each positive integer n, some set of zero or more n-ary *relation symbols*

where the word *function* and *relation* are currently simply formal add-ons to the word symbol. Symbols of the form 1-5 are called *logical symbols* while 6-8 are called *non-logical symbols*, and are dependent on the particular language.

[a] Some authors don't include this symbol as a mandatory symbol and leave it as an optional 2-ary function.

Note that we should treat this set as simply a collection of symbols: they do not yet have any *semantic* meaning, they are simply *syntax*. The arity of a function or relation is the number of variables it "takes in". So, $+$ has 2-arity, $(\_)^{-1}$ has 1-arity, 0 as 0-arity, and so forth. In terms of relation, an n-ary relation can be thought of as a n-tuple. Note too the importance of "no one of which is properly contained in another". Thinking of symbols as something that is written down, let's say for example we had a constant symbol $\triangle$ and a constant symbol $\triangle\triangle$, i.e. the second symbol contains the first. Then when we later construct sentences, the sentence $\triangle\triangle$ has the vagueness of being interpretable as either two $\triangle$ or one $\triangle\triangle$.

We will often collect the constant, functions, and relations (the constant, nouns, and verbs if you will), like so:

$$\{c_1, c_2, ..., c_n, f_1^{a_1}, f_2^{a_2}, ..., f_m^{a_m}, R_1^{b_1}, R_2^{b_2}, ..., R_l^{b_l}\}$$

where the superscript of the functions and relations represent the arity of them. Note that any collection of constants, functions, or relations, can also be infinite, and we may still denote them like in the above equation, however it is less common. The above set can be thought of as a generating set for the language, as we will shortly see. Overall, to specify a language, we will need to specify the constant, function, and relation symbols that are used.

Example 1.1: Formal Languages
When defining a language, we will take the symbols that are always part of any first-order language as implicitly part of the set so as to not clutter.

1. The langue $\mathcal{L}$ with no extra non-logical symbols is a language. Sometimes, this language is called the language of equality since $=$ is the only relationship.

2. Let's say we want to work with groups and are trying to come up with the language to work with. We know a group is a set along with a binary operation $+$ and a special symbol 0 that interacts with $+$ in a consistent way for all groups. Thus, one language for groups can be

$$\mathcal{L}_G \qquad \text{is} \qquad \{0, +\}$$

   Another possible one would be to write it in terms of multiplication:

$$\mathcal{L}_G \qquad \text{is} \qquad \{1, \cdot\}$$

   It is also reasonable to think of inverting an element to be a 1-ary operation

$$\mathcal{L}_G \qquad \text{is} \qquad \{1, (\_)^{-1}, \cdot\}$$

   Thus, we can have much flexibility when choosing a language.

3. As an example of a language with a constant, functions, and a relation: an ordered field can have the language of:

$$\mathcal{L}_F \qquad \text{is} \qquad \{0, 1, (\_)^{-1}, +, \cdot, <\}$$

4. We might think that the language of set theory has many possible symbols (ex. $\subseteq, \in, \notin, \subsetneq$, etc.). Though this is a choice, we can also define the language of set theory as simply being:

$$\mathcal{L}_{\mathbf{Set}} \qquad \text{is} \qquad \{\in\}$$

   In other words, the fundamental idea in the language of set theory is that we have containment, which is a binary relation. The use of this simplification of language will be in many proofs being easier if we only have this one [extra] symbol (similar to how computer-architecture is simpler with a reduced instruction set computer, RISC).

5. The language of field $\mathcal{L}_F$ consists of constant symbols $0, 1$ and 2 binary functions symbols, usually denoted $+$ and $\cdot$

6. The language of linear order $\mathcal{L}_O$ consists of a binary relation symbol $<$

Now that for a collection of symbols of a language, we will want to make meaningful combinations of the symbols. For example, if our language is $\{0, +, <\}$ (remember to think of this set as the generating set) and we use it to describe the order theory of the naturals (i.e. giving in that meaning), then we can reasonably say that:

$$(v_1 + 0) < v_1$$

is a sentence with meaning[1], while

$$= \exists((+ + + + 0((\forall = v_{42}$$

[1] and is has the truth value of false, more on how to gain meaning from a sentence in ref:HERE

is a meaningless combination of symbols. Thus, we will work towards finding subsets of $P(\mathcal{L})$ that have "meaning". Like with language, we will split the most common types of subsets with meanings in categories (similarly to how in english we have nouns, verbs, adjectives, and so forth). For math, the two most common categories that have meanings are *terms* and *formulas*. We define each and give examples:

Definition 1.1.2: Term

Let $\mathcal{L}$ be a language. Then a *term* of $\mathcal{L}$ is a nonempty finite string t of symbols from $\mathcal{L}$ such that that either:

1. t is a variable
2. t is a constant symbol
3. $t :\equiv ft_1t_2...t_n$ where f is any n-ary function symbol of $\mathcal{L}$ and each of the t_i is a term of $\mathcal{L}$.

The symbol $:\equiv$ is a meta-linguistic symbol that means that the symbol on the left hand side represents the same thing as on the right hand side. It is *not* part of the language $\mathcal{L}$. Note too that the 3rd point in the above definition is recursive, that is we can recursively define terms by inputting them into functions. Given countably many variable, constant symbols, and function symbols, there are only countably many terms in a language.

Example 1.2: Terms In Language

1. Let $\mathcal{L} = \{\overline{0}, \overline{1}, ..., \overline{n}, ..., +, \cdot\}$ where we have one constant symbol for each natural number, and two binary function symbols. In the following set of examples, anything with a bar over it is a term, and we are using the polish notation (which will make it easier to consistently identify a unique reading of a sentence, especially once the symbols are not familiar to us):
   (a) $\overline{714}$
   (b) $+\overline{32}$
   (c) $\cdot + \overline{123}$

   Note that $\overline{123}$ wouldn't be a term of $\mathcal{L}$, but a sequence of three terms.

Note that any term with no function symbols will be of length 1.

Overall, terms in a language $\mathcal{L}$ play the role of nouns. We will now show how to combine nouns with the "statements":

Definition 1.1.3: Formulas

Let $\mathcal{L}$ be a first-order language. Then a *formula* of $\mathcal{L}$ is a nonempty finite string φ of symbols from $\mathcal{L}$ such that either *atomic formulas*:

1. $\varphi :\equiv= t_1 t_2$ where t_1 and t_2 are terms of $\mathcal{L}$
2. $\varphi :\equiv Rt_1 t_2 ... t_n$ where R is an n-ary relation symbol of $\mathcal{L}$ and $t_1, t_2, ..., t_n$ are all terms of $\mathcal{L}$

and *recursive formulas*, or just simply *formulas*:

1. $\varphi :\equiv (\neg\alpha)$ where α is a formula of $\mathcal{L}$
2. $\varphi :\equiv (\alpha \vee \beta)$ where α and β are formulas of $\mathcal{L}$
3. $\varphi :\equiv (\forall v)(\alpha)$ where v is a variable and α is a formula of $\mathcal{L}$

If our language contains different default connectives (like $\wedge$ or $\rightarrow$), then we would have these as valid formulas (i.e. $\varphi :\equiv \alpha \wedge \beta$ or $\varphi :\equiv \alpha \rightarrow \beta$).

Be sure to realise that in the formula $(\forall v)(\alpha)$, the symbol v is a *variable*, not a term or another formula. If we have if for a formula φ, we have the string $(\forall v)(\alpha)$ as a substring, then we will say that the *scope* of the quantifier $\forall$ is α. Any symbol within α will then be said to lie within the scope of the quantifier $\forall$. Note that a formula φ may have many quantifiers, some lying within each other's scope. Note that terms are *not* formulas. Going by the linguistic analogy, terms are the nouns and the formulas are the statements. Nouns' don't have truth values, but formulas will have truth values once we get to it.

Example 1.3: Formula's In A Language

1. In the language of set theory, we may state $\exists x \forall y \neg y \in x$ (If interpreted semantically, this can be thought of stating the existence of the empty set). Can you think of what the following formula's semantic meaning is:
   (a) $\forall x \exists y \forall z \ z \in y \leftrightarrow \exists w \in x \in (z \in w)$
   (b) $\forall x \exists y \forall z \ z \in \forall w \in x (z \in w)$
   (c) $\neg x \ x \notin x$
   (d) $\forall x \forall y \exists z \forall w (w \in z \leftrightarrow w \in x \wedge w \in y)$
2. Let $\mathcal{L}_G = \{\in, *, e\}$ be the language of group. This language is an extension of the language of set. Then we may state the axioms of sets as the following terms:
   (a) $\forall a \forall \ a * b \in G$
   (b) $\forall a \forall b \forall c \ a * (b * c) = (a * b) * c$
   (c) $\exists a \forall b \ a * b = b$
   (d) $\forall a \exists b \ a * b = e$

These are the basic terms we will use. I will stress a final time to not yet interpret any symbol: $\forall$

does not yet mean "for all", $\vee$ does not yet mean "or". Assigning meaning to symbols will come in due time. Now that we have defined formula, we can see that we may choose different connectives which will imply the usual connectives we know. For example, if your connectives where $\neg$ and $\to$ (which we morally can think of as "not" and "if... then"), then we would get that

$$\begin{aligned}(\alpha \wedge \beta) &:\equiv (\neg(\alpha \to (\neg\beta)))\\ (\alpha \vee \beta) &:\equiv ((\neg\alpha) \to \beta)\\ (\alpha \leftrightarrow \beta) &:\equiv ((\alpha \to \beta) \wedge (\beta \to \alpha))\end{aligned}$$

In our case, our connectives are $\neg$ and $\vee$, thus we have:

$$\begin{aligned}(\alpha \wedge \beta) &:\equiv (\neg((\neg(\alpha) \vee (\neg\beta))))\\ (\alpha \to \beta) &:\equiv ((\neg\alpha) \vee \beta)\\ (\alpha \leftrightarrow \beta) &:\equiv ((\alpha \to \beta) \wedge (\beta \to \alpha))\\ (\exists x)(\alpha) &:\equiv (\neg(\forall x)(\neg\alpha))\end{aligned}$$

Later, we will see that the semantic meaning will correspond what is currently only a syntactic shortcut. To make life easier there are a couple of conventions we will adopt:

1. we will usually use infix notation instead of polish notation
2. we will occasionally change the size of parenthesis, or the symbol for the parenthesis, to make strings more legible
3. we will sometimes add or drop parenthesis to make strings more legible
4. If we write $\neg\alpha \to \beta$, we mean $((\neg\alpha) \to \beta)$, that is $\neg$ takes precedence
5. If we have $\alpha \to \beta \to \gamma$, we will group towards the right, meaning that could be read as $(\alpha \to (\beta \to \gamma))$

For future purposes we define the following concept

Definition 1.1.4: Subformulas

Let $\mathcal{L}$ be a language and φ a formula. Then the set of *subformulas* of φ, denoted $S(\varphi)$ is defined as follows:

1. If φ is an atomic formula, then $S(\varphi) = \{\varphi\}$
2. if φ is $(\neg\alpha)$, then $S(\varphi) = S(\alpha) \cup \{(\neg\alpha)\}$
3. If φ is $(\alpha \to \beta)$ then $S(\varphi) = S(\alpha) \cup S(\beta) \cup \{(\alpha \to \beta)\}$

It should be mentioned that any formula can only have one way of being read: there cannot be confusion with whether a symbol is a symbol or not:

Theorem 1.1.1: Unique Readability

Let $\mathcal{L}$ be a language and let α be a formula of $\mathcal{L}$. Then α can only satisfy one of the condition's of the definition of a formula, or if it satisfies none it cannot be a formula.

***Proof* :**
The proof was left as an exercise.

For future reference, it is important to think of extensions of language:

Definition 1.1.5: Language Extension

Let $\mathcal{L}$ and $\mathcal{L}'$ be first order languages. Then $\mathcal{L}'$ is an *extension* of $\mathcal{L}$ if $\mathcal{L} \subseteq \mathcal{L}'$, that is if every non-logical symbol of $\mathcal{L}$ is a non-logical symbol of the same kind in $\mathcal{L}'$.

As an example, when we defined group with $\in$, that language was an extension of $\mathcal{L}_{\mathbf{Set}}$.

1.2 Sentences

Given a language $\mathcal{L}$, we can create formulas. The formulas we are most interested in are those which require no choice, or assignment of value. For example, if I write $\forall x\ x = y$, once we define how to give semantic meaning to this formula, we will see that the "value" of the formula depends on the choice of y. However, if we write $\forall x \forall y\ x = y$ or $\forall x \exists y\ x = y$ then y is no bound to the given logic. A formula with no "free" variables will be called a sentence.

Before continuing to define these concepts more rigorously, we will make a special definition for the language of number theory as it is extremely valuable for future discussions:

Definition 1.2.1: Number Theory Language

Let $\mathcal{L}_{NT}$ be the language generated by $\{0, S, +, \cdot, E, <\}$, where 0 is a constant, S is a unary function (the "successor function"), $+$, $\cdot$, and E are binary function (E for exponential) and $<$ is a binary relation. Then we usually call $\mathcal{L}_{NT}$ the language of *number theory*

This language will become important for us later, but for now let's just use it to demonstrate the idea of a sentence. Take the following two formulas[2]:

$$\neg(\forall x)[(y < x) \vee (y = x)] \tag{1.1}$$
$$(\forall x)(\forall y)[(x < y) \vee (x = y) \vee (x < y)] \tag{1.2}$$

The first sentence reads "not for all x is y less than or equal to x", while the second sentence is the trichotomy law of $<$. The second formula is true about the collection of natural numbers as we already understand them. The first formula has the problem of y not being quantities: are we picking particular value of y, any value of y, all possible values of y? In other words, y is a *free variable*. We may choose what y to put into that formula. Think of integration:

$$F(x) = \int_0^x f(t)dt$$

In this case, the value of x is the free variable because we are free to choose what it is. This makes a formula dependent upon a choice, and hence not holding an inherent truth value. We will give variables that bring in an element of choice into a formula a name:

[2]Notice that we are already abusing notation here

Definition 1.2.2: free variable

Let v be a variable and φ a formula. We will say that v is *free* in φ if:

1. φ is an atomic formula and v occurs in (as a symbol in) φ
2. $\varphi :\equiv (\neg\alpha)$ and v is free in α
3. $\varphi :\equiv (\alpha \vee \beta)$ and v is free in at least one of α or β
4. $\varphi :\equiv (\forall u)(\alpha)$ and v is not u and v is free in α

Example 1.4: Free Variables

1. In the formula
$$\forall v_2 \neg(\forall v_3)(v_1 = S(v_2) \vee v_3 = v_2)$$
the variable v_1 is free whereas the variables v_2 and v_3 are not free
2. A more subtle example is the following
$$(\forall v_1 \forall v_2(v_1 + v_2 = 0)) \vee v_1 = S(0)$$
Notice that v_1 here is free while v_2 is not free; in other words, be weary of parenthesis and the scope of quantifiers in parenthesis.
3. An example of a variable being both free and not free, depending on the scope, is v_6 in the following formula:
$$\forall v_0(= v_1 f_3^1 v_3 \to (\neg\forall v_3 P_9^1 v_3))$$

Definition 1.2.3: Sentence

Let $\mathcal{L}$ a language. Then a *sentence* in $\mathcal{L}$ is a formula of $\mathcal{L}$ that contains no free variables.

Example 1.5: Sentences

The following uses $\mathcal{L}_{NT}$ as a language

1. the formula $1 + 1 = 2$ is a sentence
2. the formula $(\forall x)(x + 1 = x)$ is a sentence

As a rule of thumb, if it seems like you can determine if a statement is true or false without any choice then it's a sentence. Intuitively, the first statement is true while the second statement is false. We will explore exactly what is the nature of this in section 1.4.

1.3 Structure

It was stressed many times that the symbols in a language do not come with any meaning attached to them. The goal of this section is to show how we attach a particular meaning to the symbols

Definition 1.3.1: Structure (Model)

Let $\mathcal{L}$ be a language. Then an $\mathcal{L}$-structure $\mathfrak{A}$ is a nonempty set A, called the *universe of* $\mathfrak{A}$ together with:

1. For each constant symbol c of $\mathcal{L}$, an element $c^{\mathfrak{A}}$ in A
2. For each n-ary function symbol f of $\mathcal{L}$, a function $f^{\mathfrak{A}} : A^N \to A$
3. For each n-ary relation symbol R of $\mathcal{L}$, an n-ary relation $R^{\mathfrak{A}}$ on A (i.e. a subset of A^n)

Often, we write $|\mathfrak{A}| := A$.

Note that all functions will be total functions (the entire domain maps to something in the co-domain) unless otherwise specified. Note too that $c^{\mathfrak{A}}$ need not be the same symbol as c: for example, c might be \$ while $c^{\mathfrak{A}}$ is "Friends". Often, a structure will be written as an ordered pair, for example:

$$\mathfrak{A} = (A, c_1^{\mathfrak{A}}, c_2^{\mathfrak{A}}, f_1^{\mathfrak{A}}, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}})$$

Naturally, we will immediately start simplifying notation as long as it's not ambiguous. For example, the standard structure on $\mathcal{L}_{NT}$ will be denoted

$$\mathfrak{N} = (\mathbb{N}, 0, S, +, \cdot, E, <)$$

where we have dropped the superscript and take $\mathbb{N} := \{0, 1, 2, ...\}$. In set theory, 0 is usually considered a natural number. Naturally, we can construct different $\mathcal{L}_{NT}$-structures, for example we can just as well construct a (mod 2) structure with the language $\mathcal{L}_{NT}$, and label this $\mathcal{L}_{NT}$-structure $\mathfrak{T}$.

Example 1.6: Structure

1. $\mathfrak{Q} = (\mathbb{Q}, <)$ is a structure with the language of linear order on $\mathbb{Q}$. Other structures with the language $<$ maybe be $(\mathbb{R}, <)$, $((\{0\}, \emptyset), <)$, $((\mathbb{N}, \mathbb{N}), <)$ are other examples, given appropriate definition of $<$.
2. Let $\mathcal{L}_F$ be the language of fields we defined earlier. Then we may certainly make a structure for $\mathcal{L}_F$ that will be a field, however there is no restriction to make it follows the axioms of a field: we may also map the two field operations to the same function and have a group structure, or really almost any other structures.
3. An important structure that will be useful to us later on is the *Henkin structure* of a language. Let $\mathcal{L}$ be $\{0, f, g, R\}$, with constant symbol 0, unary function symbol f, binary function symbol g, and 3-ary relation symbol R. We now define a $\mathcal{L}$ structure $\mathfrak{B}$: the universe, B, is the set of all variable-free $\mathcal{L}$-terms; the universe contains all terms that do not have a variable. The constant $0^{\mathfrak{B}}$ is the term 0. The functions $f^{\mathfrak{B}}$ and $g^{\mathfrak{B}}$ are defined by concatenation like so:

$$f^{\mathfrak{B}}(t) = ft \qquad g^{\mathfrak{B}}(t, s) = gts$$

Note that the function symbols f, g are indeed variable free, and hence this is a valid function. Finally, we may let the relation $R^{\mathfrak{B}}$ be anything we want. As a random example:

$$R^{\mathfrak{B}} = \{(r, s, t \in B^3) \mid \text{ the number of function symbols in } r \text{ is even}\}$$

This structure on $\mathcal{L}$ will be important to us when we look at the completeness theorem in chapter ref:HERE.

1.4 Truth in a Structure

We know get to assigning truth value's to sentences. We will do so by having an assignment function. There are many assignment functions we will use at varying levels: we will start by having a *variable* assignment function which gives to each variable in $Vars$ an element of A. We then build on this by defining a *term assignment function*, which uses the variable assignment function and then also allows us to give to each term of $\mathcal{L}$ an element of A. Finally, we will link formulas to an assignment function

Definition 1.4.1: Variable Assignment Function

Let $\mathcal{L}$ be a language and $\mathfrak{A}$ be a $\mathcal{L}$-structure. Then a *variable assignment function* into $\mathfrak{A}$ is a function s that assigned to each variable an element of the universe A. So a variable assignment function into $\mathfrak{A}$ is any function with domain $Vars$ and codomain A, $s : Vars \to A$.

Note variable assignment function need not be injective or surjective.

Example 1.7: Variable Assignment Function

Take $\mathcal{L}_{NT}$ with the standard structure $\mathfrak{N}$. Then the function s mapping $s(v_i) = i$ is a variable assignment function. Another example would be s' where

$$s'(v_i) = \text{the smallest prime number that does not divide } i$$

Definition 1.4.2: Modification Of The Assignment Function

Let s be a variable assignment function into $\mathfrak{A}$, x is a variable, and $a \in A$. Then $s[x|a]$ is the variable assignment function into $\mathfrak{A}$ defined as:

$$s[x|a](v) = \begin{cases} s(v) & \text{if } v \text{ is a variable other than } x \\ a & \text{if } v \text{ is the variable } x \end{cases}$$

We call the function $s[x|a]$ an *x-modification of the assignment function s.*

In other words, an x-modification of an assignment function overrides what $s(x)$ is; it force it to map to a. Next, we will extend the variable assignment function s to a term assignment function $\overline{s}$. This function will assign an element of the universe to each term of the language $\mathcal{L}$, and is "auto-generated" based off of the variable-assignment function and the functions of a given $\mathcal{L}$-structure:

Definition 1.4.3: Term Assignment Function

Let $\mathcal{L}$ be a language, $\mathfrak{A}$ an $\mathcal{L}$-structure, and s a variable assignment function into $\mathfrak{A}$. Then the function $\overline{s}$, called the *term assignment function generated by* s, is the function with domain consisting of the set of $\mathcal{L}$-terms and codomain A defined recursively as follows:

1. if t is a variables, $\overline{s}(t) = s(t)$
2. If $t :\equiv c$ is a constant symbol, then $\overline{s}(t) = c^{\mathfrak{A}}$
3. if $t :\equiv ft_1t_2...t_n$, then $\overline{s}(t) = f^{\mathfrak{A}}(\overline{s}(t_1), \overline{s}(t_2), ..., \overline{s}(t_n))$

We will mostly be interested in truth of sentences, but for future purposes we will first describe truth (or "satisfaction") for arbitrary formulas, relative to an assignment function

Definition 1.4.4: Structure Satisfying Assignment Formula ($\mathfrak{M} \models \varphi[s]$)

Let $\mathfrak{A}$ be an $\mathcal{L}$-structure, φ an $\mathcal{L}$-formula, and $s : Vars \to A$ an assignment function. Then we will say that $\mathfrak{A}$ *satisfies* φ *with assignment* s, and write $\mathfrak{A} \models \varphi[s]$ in the following circumstances:

1. Atomic formulas:
   (a) if $\varphi :\equiv= t_1t_2$ and $\overline{s}(t_i)$ is the same element of the universe A as $\overline{s}(t_2)$ (i.e. $\overline{s}(t_1) = a = \overline{s}(t_2)$)
   (b) If $\varphi :\equiv Rt_1t_2...t_n$ and $(\overline{s}(t_1), \overline{s}(t_2), ..., \overline{s}(t_n)) \in R^{\mathfrak{A}}$
2. recursive formulas:
   (a) $\varphi :\equiv (\neg\alpha)$ and $\mathfrak{A} \not\models \alpha[s]$ (where $\not\models$ means "does not satisfy")
   (b) if $\varphi :\equiv (\alpha \vee \beta)$ and $\mathfrak{A} \models \alpha[s]$ or $\mathfrak{A} \models \beta[s]$, or both
   (c) If $\varphi :\equiv (\forall x)(\alpha)$ and, for each a of A, $\mathfrak{A} \models \alpha[s(x|a)]$

If Γ is a set of $\mathcal{L}$-formulas, we say that $\mathfrak{A}$ satisfies Γ with assignment s if for each $\gamma \in \Gamma$, $\mathfrak{A} \models \gamma[s]$, and write $\mathfrak{A} \models \Gamma[s]$. We say that Γ is *satisfiable with* s if there exists a model $\mathfrak{M}$ that satisfies Γ.

Note that the symbol $\models$ is *not* part of the language $\mathcal{L}$. Instead, $\models$ is a metalinguistic symbol that we use to talk about formulas in the language and structures of the language. Notice too that we finally have our usual interpretation of $\neg$ as negative, $\vee$ as or, and $\forall$ as for all. In some first-order languages, the $\vee$ is replaced by $\to$, in which case we would get if $\varphi :\equiv \alpha \to \beta$ and $\mathfrak{M} \models \varphi[s]$ if

$$\mathfrak{M} \models \beta[s] \text{ whenever } \mathfrak{M} \models \alpha[s]$$

Example 1.8: Truth Assignment

1. If $\mathcal{L} = \{\}$ is the empty language, $\mathfrak{A} = (\{1, 2\})$ a $\mathcal{L}$-structure, then take $s(v_1) = 1$ and $s(v_2) = 2$. Let $\varphi :\equiv v_1 = v_2$. Then $\mathfrak{A} \not\models \varphi[s]$, that is $\mathfrak{A}$ satisfies φ with s. On the other hand, if $s(v_2) = 2$, then $\mathfrak{A} \models \varphi[s]$. It should be easy to come up with other assignment functions s and or other $\mathcal{L}$-structures that will give the formula $v_1 = v_2$ (or in formal notation $= v_1v_2$) a different true value.

We have determined whether a general formula satisfies a $\mathcal{L}$-structure given an assignment function, but what about the special case of sentences? Since they don't have free variables, they should in a sense be independent of any choice function. That is indeed the case. For now, we will take the choice of assignment function for sentences too, and show that the choice of assignment function is inconsequential. From there, we will conclude that σ is a true sentence in $\mathfrak{A}$ if and only if $\mathfrak{A} \models \sigma[s]$ for all possible variable assignment function s.

Lemma 1.4.1: Equal On Variables Then Equal On Terms

Let s_1, s_2 be variable-assignment functions into a structure $\mathfrak{A}$ such that $s_1 = s_2$, that is for all variables:

$$s_1(v) = s_2(v)$$

Then $\overline{s_1} = \overline{s_2}$, that is for all variables:

$$\overline{s_1}(v) = \overline{s_2}(v)$$

***Proof* :**
We do structural induction on the terms. If t is a variable or a constant symbol, the result is immediate. Arguing inductively, if $t :\equiv ft_1t_2...t_n$ then since $\overline{s_1}(t_i) = \overline{s_2}(t_i)$ for all $1 \leq i \leq n$ by the induction hypothesis, the definition of $\overline{s_1}(t)$ and $\overline{s_2}(t)$ are identical, and so by the principle of induction

$$\overline{s_1} = \overline{s_2}$$

as we sought to show.

Proposition 1.4.1: Agree On Variables Then Models The Same

Let s_1, s_2 be variables assignment functions into a structure $\mathfrak{A}$ such that $s_1(v) = s_2(v)$ for every *free variable* v in the formula φ. Then

$$\mathfrak{A} \models \varphi[s_1] \qquad \text{iff} \qquad \mathfrak{A} \models \varphi[s_2]$$

***Proof* :**
We again use structural induction. If $\varphi :\equiv t_1 = t_2$ then the free variable of φ are exactly the variables that occur in φ. Thus, by lemma 1.4.1 we have $\overline{s_1}(t_1) = \overline{s_2}(t_1)$ and $\overline{s_1}(t_2) = \overline{s_2}(t_2)$ meaning they are the same element of the universe A, and so $\mathfrak{A} \models (t_1 = t_2)[s_1]$ if and only if $\mathfrak{A} \models (t_1 = t_2)[s_2]$. Other base case of $\varphi :\equiv Rt_1t_2...t_n$ is similar.

For the inductive case, we'll show the $\varphi :\equiv \neg\alpha$ and leave the $\alpha \vee \beta$ as an exercise. Notice that $\neg\alpha$ and α have exactly the same free-variables, so s_1 and s_2 agree on the free-variables of α. Thus, by the inductive hypothesis $\mathfrak{A} \models \varphi[s_1]$ if and only if $\mathfrak{A} \models \varphi[s_2]$.

Finally, for $\varphi :\equiv (\forall x)(\alpha)$, the only variable that might be free in α that is not free in $(\forall x)(\alpha)$ is x. Thus, $s_1[x|a]$ and $s_2[x|a]$ agree on all free variables of α, and so by the induction hypothesis for each $a \in \mathfrak{A}$, $\mathfrak{A} \models \alpha[s_1[x|a]]$ if and only if $\mathfrak{A} \models \alpha[s_2[x|a]]$. Thus, by of a variable assignment function,

$$\mathfrak{A} \models \varphi[s_1] \qquad \text{if and only if} \qquad \mathfrak{A} \models \varphi[s_2]$$

Thus, by the principle of induction, this hols true for all of φ, completing the proof.

The following corollary is important as it shows that sentences are "independent" of variable assignment functions

Corollary 1.4.1: Satisfiability For Sentences

Let σ be a sentence in the language $\mathcal{L}$ and $\mathfrak{A}$ an $\mathcal{L}$-structure. Then $\mathfrak{A} \models \sigma[s]$ for all assignment functions s, or $\mathfrak{A} \models \sigma[s]$ for no assignment functions s

***Proof* :**
By definition, σ has *no* free variables. And so if s_1 and s_2 are two assignment functions, they (vacuously) agree on free variables. Thus, $\mathfrak{A} \models \sigma[s_1]$ if and only if $\mathfrak{A} \models \sigma[s_2]$. Since this is true in general, we see that either $\mathfrak{A} \models \sigma[s]$ for all assignments or no assignments, as we sought to show.

A consequence of the above corollary is that if σ is a sentence, $\mathfrak{M} \models \sigma$ if and only if there is some assignment $v : V \to |\mathfrak{M}|$ such that $\mathfrak{M} \models \sigma[s]$.

Definition 1.4.5: Modelled Formulas and True Sentences ($\mathfrak{M} \models \varphi$)

Let φ be a formula in a language $\mathcal{L}$ and $\mathfrak{A}$ be a $\mathcal{L}$-structure. Then

1. we say that $\mathfrak{A}$ is a *model* for φ if and only if $\mathfrak{A} \models \varphi[s]$ for every assignment function s, and we write $\mathfrak{A} \models \varphi$.
2. If Φ is a set of $\mathcal{L}$-formulas, we will say that $\mathfrak{A}$ models Φ, and write $\mathfrak{A} \models \Phi$ if and only if $\mathfrak{A} \models \varphi$ for each $\varphi \in \Phi$.
3. We say that φ or Φ is *satisfiable* if there exists a model $\mathfrak{M}$ that satisfies it (i.e. a model $\mathfrak{M}$ such that Γ is satisfiable for every assignment function s).
4. For the case of φ being a *sentence*, then we have $\mathfrak{A} \models \sigma$ if and only if $\mathfrak{A} \models \sigma[s]$ for any assignment function s. In this case we will say that the sentence φ is *true* in $\mathfrak{A}$.

Example 1.9: True Sentences
Let's take the number theory structure with language $\mathcal{L}_{NT}$ and standard structure

$$\mathfrak{N} = (\mathbb{N}, 0, S, +, \cdot, E, <)$$

Let s be the variable assignment function $s(v_i) = 2i$ (ex. $s(3) = 6$). As an example of the term-assignment function that comes from it:

$$\begin{aligned} \overline{s}(v_1 + v_1) \quad &\text{is} \quad +^{\mathfrak{N}}(\overline{s}(v_1), \overline{s}(v_1)) \\ &\text{is} \quad +^{\mathfrak{N}}(2, 2) \\ &\text{is} \quad 4 \end{aligned}$$

or similarly:

$$\begin{aligned} \overline{s}(SSSS0) \quad &\text{is} \quad S^{\mathfrak{N}}(S^{\mathfrak{N}}(S^{\mathfrak{N}}(S^{\mathfrak{N}}(0)))) \\ &\text{is} \quad 4 \end{aligned}$$

Let's now look for true sentences. Consider the sentence:

$$(\forall v_1)\neg(\forall v_2)\neg(v_1 = v_2 + v_2)$$

where $\neg(\forall v_2)\neg$ is usually shorthanded to $(\exists v_2)$. Parsing this into a english, we see that the sentence is asking whether all numbers are the sum of two identical numbers; if all numbers are even. This is intuitively wrong, so let's start parsing the sentence by the rules we've been given to come to the same conclusion:

$$\begin{aligned} \mathfrak{A} \models \sigma[s] \quad &\text{iff} \quad \text{For every } a \in \mathbb{N},\ \mathfrak{A} \models \neg(\forall v_2)\neg(v_1 = v_2 + v_2)s[v_1|a] \\ &\text{iff} \quad \text{For every } a \in \mathbb{N},\ \mathfrak{A} \not\models (\forall v_2)\neg(v_1 = v_2 + v_2)s[v_1|a] \\ &\text{iff} \quad \text{For every } a \in \mathbb{N}, \text{there exists a } b \in \mathbb{N},\ \mathfrak{A}(v_1 = v_2 + v_2)s[v_1|a][v_2|b] \end{aligned}$$

we many now plug in a and see that not b will satisfy this property, showing that $\mathfrak{A} \not\models \sigma[s]$ we many now plug in a and see that not b will satisfy this property, showing that $\mathfrak{A} \not\models \sigma[s]$. Thus, we may say that $\mathfrak{A}$ does not model σ, and so σ is *false*.

Proposition 1.4.2: Existential Quantifier

Let $\mathfrak{M}$ be a structure of a first order language$\mathcal{L}$, s an assignment for $\mathfrak{M}$, x a variable, and φ a formula of $\mathcal{L}$. Then $\mathfrak{M} \models \exists x, \varphi[s]$ if and only if $\mathfrak{M} \models \varphi[s(x|m)]$ for some $m \in |\mathfrak{M}|$

***Proof* :**
This is unwinding the definitions and using our usual notion of logic.

Proposition 1.4.3: Important Shorthand's

Let $\mathfrak{M}$ be a structure for $\mathcal{L}$ and α, β be sentences of $\mathcal{L}$. Then:

1. $\mathfrak{M} \models \neg\alpha$ if and only if $\mathfrak{M} \not\models \alpha$
2. $\mathfrak{M} \models \alpha \to \beta$ if and only if $\mathfrak{M} \models \beta$ whenever $\mathfrak{M} \models \alpha$
3. $\mathfrak{M} \models \alpha \vee \beta$ if and only if $\mathfrak{M} \models \alpha$ or $\mathfrak{M} \models \beta$
4. $\mathfrak{M} \models \alpha \wedge \beta$ if and only if $\mathfrak{M} \models \alpha$ and $\mathfrak{M} \models \beta$
5. $\mathfrak{M} \models \alpha \leftrightarrow \beta$ if and only if $\mathfrak{M} \models \alpha$ exactly when $\mathfrak{M} \models \beta$
6. $\mathfrak{M} \models \forall x\alpha$ if and only if $\mathfrak{M} \models \alpha$

Similarly, if Σ is a set of formula, ψ and ρ are some formulas, then

$$\Sigma \cup \{\psi\} \models \rho \qquad \text{if and only if} \qquad \Sigma \models (\psi \to \rho)$$

Proof **:**
easy exercise

Note that α and β are sentences in the above. If they were formulas, we must be weary of particular assignment functions.

1.5 Substitutions and Substitutability

we now tackle the problem of when we can substitute a variable by a term. For example, if we have a term $(\forall x)\varphi(x)$ which is true in a particular structure $\mathfrak{A}$, we see that if we replace the variable x by the constant c (recall that $c^{\mathfrak{A}}$ in a structure is an element of A), then we would expect $\varphi(c)$ will also be true. However, things are not always so simple. Take for example $\mathfrak{A} \models \forall x \exists y \neg(x = y)$. This sentence is true in any structure $\mathfrak{A}$ if A has at least 2 elements. Then if we replace the variable x with the variable u, then we would expect that the truth values stays the same. However, if we replace x with y, then we see that the sentence is no longer true, for any structure! In other words, our bad choice of substitution changed the truth value of the formula. We will thus be more careful with how we proceed with substituting terms inside a quantifier that binds a variable involved with the term

Definition 1.5.1: Term Substitution

let u be a term, x a variable, and t a term. Then we define the term u_t^x (read "u with x replaced by t") as follows:

1. If u is a variable not equal to x, then u_t^x is u
2. if u is x, then u_t^x is t
3. if u is a constant symbol, then u_t^x is u
4. if $u :\equiv fu_1u_2...u_n$, where f is an n-ary function symbol and the u_i are terms, then[a]

$$u_t^x \text{ is } f(u_1)_t^x(u_2)_t^x...(u_n)_t^x$$

[a]The parenthesis technically shouldn't be there, but they were added for readability

Essentially, all this definition is saying is that we should find the variable x (if it's in the term u) and replace it with t.

Example 1.10: Term Substitution

Let's say u is $f(x,y)+h(z,x,g(x))$, and we want to replace x with t which is $g(c)$. Processing the definition, we get u_t^x is:

$$f(g(c),y)+h(z,g(c),g(g(c)))$$

We define it similarly for a formula

Definition 1.5.2: Formula Substitution

Suppose that φ is an $\mathcal{L}$-formula, t is a term, and x is a variable. Then we define the formula φ_t^x (read "φ with x replaced by t") as:

1. if $\varphi :\equiv u_1 = u_2$ then φ_t^x is $(u_1)_t^x = (u_2)_t^x$
2. If $\varphi :\equiv Ru_1u_2...u_n$, then φ_t^x is $R(u_1)_t^x(u_2)_t^x...(u_n)_t^x$
3. if $\varphi :\equiv \neg(\alpha)$ then φ_t^x is $\neg(\alpha_t^x)$
4. If $\varphi :\equiv (\alpha \vee \beta)$, then φ_t^x is $(\alpha_t^x \vee \beta_t^x)$
5. If $\varphi :\equiv (\forall y)(\alpha)$, then

$$\varphi_t^x = \begin{cases} \varphi & \text{if } x \text{ is } y \\ (\forall y)(\alpha_t^x) & \text{otherwise} \end{cases}$$

Example 1.11: Formula Substitution

Suppose that we have the formula φ

$$P(x,y) \to [(\forall x)(Q(g(x),z)) \vee (\forall y)(R(x,h(x)))]$$

then if t is the term $g(c)$, we get that φ_t^x is

$$P(g(c),y) \to [(\forall x)(Q(g(g(c)),z)) \vee (\forall y)(R(g(c),h(g(c))))]$$

With the preliminary definitions done, we can now talk about when we can substitute a term for a variable without running into the problems mentioned at the beginning of this section; we will not substitute a term in such a way that a variable contained in that term is inadvertently bound by a quantifier

Definition 1.5.3: Substitutable

Let φ be an $\mathcal{L}$-formula, t a term, and x a variable. Then we say that t is *substitutable* for x in φ, or x can be *substituted* by t, if:

1. φ is atomic
2. $\varphi :\equiv \neg(\alpha)$ and t is substitutable for x in α
3. $\varphi :\equiv (\alpha \vee \beta)$ and t is substitutable for x in both α and β
4. $\varphi :\equiv (\forall y)(\alpha)$ and either
   (a) x is not free in φ (i.e. x is bound or does not exist in φ)
   (b) y does not occur in t and t is substitutable for x in α

Note that φ_t^x is defined whether or not t is substitutable for x in φ. Usually we will want to do substitutions only if the terms are substitutable, but we can do a substitution even if it will change the truth value. On the other hand, we will often only substitute if it is substitutable.

Example 1.12: Substitutability

1. x is always substitutable for x in any formula φ, namely φ_x^x is simply φ
2. Let $\varphi :\equiv \forall y\ x = y$. Then Then y is not substitutable by x, since the new y would be "bounded" by the quantifier $\forall y$

1.6 Logical Implication

The last bit of language we will gain is that of formula implication. In mathematics, it is very common to have answers to questions of the form: If these statements are true, it is necessarily the case that the other statement(s) is/are true.

Definition 1.6.1: Logical Implication ($\Delta \models \Gamma$)

Suppose that Δ and Γ are sets of $\mathcal{L}$-formulas. Then we will say that Δ *logically implies* Γ and write $\Delta \models \Gamma$ if for every $\mathcal{L}$-structure $\mathfrak{A}$, if $\mathfrak{A} \models \Delta$, then $\mathfrak{A} \models \Gamma$

To be pedantic, this definition does not necessarily mean that there is a causal relation between Δ and Γ, it means that it so happens that if Δ is true, it must be that Γ is true. Notice too that it is independent of any $\mathcal{L}$-structure, meaning the implication should be "universally" valid.

Example 1.13: Logical Implication

1. Let $\mathfrak{M}$ be any structure. Then $\mathfrak{M} \models \{\varphi\}$ if and only if $\mathfrak{M} \models \varphi$, then $\{\varphi\} \models \varphi$
2. Let α, β be formulas. Show that $\{\alpha \to \beta, \alpha\} \models \beta$

Definition 1.6.2: Valid Formula (Tautology and Contradiction) ($\models \varphi$)

Let φ be an $\mathcal{L}$-formula. Then we say that φ is valid (or that it is a tautology) if $\emptyset \models \varphi$, in other words φ is true in every $\mathcal{L}$-structure with every assignment function s. If $\models \neg\varphi$, we say that φ is a *contradiction*.

Example 1.14: Valid Formula and Models

1. $\models \forall x\ x = x$ is a true sentence in all models and in all term assignment functions. On the other hand $\exists x\ \neg x = x$ is a contradiction
2. $\models \varphi \to \varphi$ is a tautology, while $\models \neg(\varphi \to \varphi)$ is a contradiction.

Proposition 1.6.1: Modelling And Contradiction

Suppose φ is a contradiction. Then $\mathfrak{M} \models \varphi[s]$ is false for every structure $\mathfrak{M}$ and assignment s. As a consequence, Σ is satisfiable if and only if there is no contradiction χ such that $\Sigma \models \chi$.

Proof **:**
exercise

Finally, we require the following terminology for later

Definition 1.6.3: Theory and Axioms

If Σ is a set of sentences then the theory of Σ is

$$\mathrm{Th}(\Sigma) = \{\tau \mid \tau \text{ is a sentence and } \Sigma \models \tau\}$$

If Δ is a set of sentences and $\mathcal{S}$ is a collection of structures, then Δ Is a set of (non-logical) axioms for $\mathcal{S}$ if for every structure $\mathfrak{M}$, $\mathcal{S}$ iff $\mathfrak{M} \models \Delta$

Example 1.15: Theory And Axiom

1. example of theory
2. Let $\{\exists x \exists y((\neg(x = y))\forall z(z = x \vee z = y))\}$. Every structure over the language $\mathcal{L} = \{\}$ satisfying this sentence must have exactly two elements in the universe, and so the above set is a set of non-logical axioms for the collection of sets of cardinality 2.

1.7 *Propositional Language

We will take a moment to focus on another type language. This language will be purely focused on determining where a given formula φ is either *True* or *False*. The language will feel restrictive as compared to first-order language, and it indeed has limitations: if not careful with our definition of atomic formulas we may raise a Russel-like contradiction. However, its simplicity also has it's advantages: this language will capture the intuition that a given statement can either be assignment *true* (T) or *false* (F). Given a model $\mathcal{M}$ with a language $\mathcal{L}$, we may convert many first-order questions into propositional questions. This will come in handy in the next chapter where we will link "deductibility" (the ability to prove or deduce a statement) to "satisfiability" ($\mathcal{M} \models \varphi$). Due to this link, one of my professors called propositional logic "0th order logic".

Definition 1.7.1: Propositional Language

The symbols of $\mathcal{L}_P$ are

1. Parenthesis: (and)
2. Connectives: $\neg$ and $\rightarrow$
3. Atomic formulas $A_0, A_1, ..., A_n, ...$

A *formula* of $\mathcal{L}_P$ are finite sequences or strings of symbols from $\mathcal{L}_P$ satisfying one of the following rules:

1. Every atomic formula is a formula
2. If α is a formula, $(\neg\alpha)$ is a formula
3. if α and β are formulas, $\alpha \rightarrow \beta$ is a formula
4. No other sequence of symbols is a formula

Like first-order languages, we have shorthand's:

1. $(\alpha \wedge \beta)$ is $(\neg(\alpha \rightarrow (\neg\beta)))$
2. $(\alpha \vee \beta)$ is $((\neg\alpha) \rightarrow \beta)$
3. $(\alpha \leftrightarrow \beta)$ for $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

Definition 1.7.2: Propositional Sub formula

Let φ be a formula of $\mathcal{L}_P$. Then the subformulas of φ, $S(\varphi)$ is defined as follows:

1. if φ is an atomic formula, $S(\varphi) = \{\varphi\}$
2. if φ is $(\neg\alpha)$ then $S(\varphi) = S(\alpha) \cup \{(\neg\alpha)\}$
3. If φ is $(\alpha \rightarrow \beta)$ then $S(\varphi)$ is $S(\alpha) \cup S(\beta) \cup \{(\alpha \rightarrow \beta)|\}$

Just like for first order language, we have unique readability for propositional language. Instead of having structure and a variable-assignment function, we define a truth assignment function directly on the formulas:

Definition 1.7.3: Truth Assignment Function

A *truth assignment* is a function v whose domain is the set of all formulas of $\mathcal{L}_P$ and whose range is any binary set, say $\{T, F\}$ such that

1. $v(A_n)$ is defined for every atomic formula A_n
2. For any formula α,
$$v((\neg\alpha)) = \begin{cases} T & \text{if } v(\alpha) = F \\ F & \text{if } v(\alpha) = T \end{cases}$$
3. For any α, β:
$$v((\alpha \to \beta)) = \begin{cases} F & \text{if } v(\alpha) = T \text{ and } v(\beta) = F \\ T & \text{Otherwise} \end{cases}$$

It is easy to show that if δ is any formula and u, v are truth assignments such that $u(A_n) = v(A_n)$ for all atomic formulas which occur in δ, then $u(\delta) = v(\delta)$.

Proposition 1.7.1: Relating Our Shorthand With Our Intuition

Let α, β be formulas and v a truth assignment function. Then:

1. $v(\neg\alpha) = T$ if and only if $v(\alpha) = F$
2. $v(\alpha \to \beta) = T$ if and only if $v(\beta) = T$ whenever $v(\alpha) = T$
3. $v(\alpha \wedge \beta) = T$ if and only if $v(\alpha) = T$ and $v(\beta) = T$
4. $v(\alpha \vee \beta) = T$ if and only if $v(\alpha) = T$ or $v(\beta) = T$
5. $v(\alpha \leftrightarrow \beta) = T$ if and only if $v(\alpha) = v(\beta)$

***Proof* :**
These are easy to see if we break down the problem into truth tables.

Note that $\to$ is *not* associative, that is $\alpha \to (\beta \to \gamma)$ doesn't model $(\alpha \to \beta) \to \gamma$, and vice-versa.

Definition 1.7.4: Satisfied Formula

Let v be a truth assignment and φ a formula. Then we say that v *satisfies* φ if $v(\varphi) = T$. If Σ is a set of formulas, we will say v satisfies Σ if $v(\sigma) = T$ for all $\sigma \in \Sigma$.

Definition 1.7.5: Satisfiable, Tautology, Contradiction

We say that φ (resp. Σ) is *satisfiable* if there is a truth assignment such that $v(\varphi) = T$ (or $v(\sigma) = T$ for all Σ), is a *tautology* if it is satisfied by every truth assignment, and a *contradiction* if there is not truth assignment satisfying it

A natural tautology is $\alpha \to \alpha$ and a natural contradiction is $\neg(\alpha \to \alpha)$. β More generally, β a tautology if and only if $\neg\beta$ is a contradiction.

Definition 1.7.6: Propositional Implication

A set of formulas Σ *implies* a formula φ if every truth assignment v which satisfies Σ also satisfies φ. We write $\Sigma \models \varphi$, and $\Sigma \not\models \varphi$ in the opposite case. If Σ is empty, we usually write $\models \varphi$ instead of $\emptyset \models \varphi$.
If Δ and Γ are sets of formulas, then Δ implies Γ if every truth assignment v which satisfies Δ also satisfies Γ

Proposition 1.7.2: Properties of Propositional Implication

1. If $\Gamma \subseteq \Sigma$ is a subset of formulas then $\Sigma \models \Gamma$
2. $\Sigma \cup \{\psi\} \models \rho$ if and only if $\Sigma \models \psi \to \rho$
3. Σ is satisfiable if and only if there is no contradiction χ such that $\Sigma \models \chi$

***Proof* :**
exercise

1.8 Summary

1. **terms and formulas**: terms are the nouns which can either be a variable, a constant, or a function taking in a term. A formula can either be $t_1 = t_2$ for some two terms t_1, t_2, some n-relation $Rt_1...t_n$, or recursively built to add negations, or's, or foralls on a variable: $\neg\varphi$, $\alpha \vee \beta$, $(\forall v)(\varphi)$
2. **infix notation and shortcuts**:
3. **subformulas**: recursively go through and compile all the subformulas within a formula.
4. **free variables**: within a formula, we may have a variable v_i which is not quantified by a $\forall$.
5. **sentence**: a formula with no free variables.
6. **structure (model)**: a set A (called the universe) along with a collection of functions, relations, and a subset of A which be used for associated functions to function symbols, relations to relations symbols, and constants to constant symbols.

7. **variable and term assignment function**: If $Vars \subseteq \mathcal{L}$ is the collection of all variables, then $s : Vars \to A$ is a variable assignment function. This can then be extended to be a term assignment function. We can also override a particular assignment by doing $s[x|a](v)$, which will on input of variable x give a instead of $s(x)$.

8. **Modelling** ($\mathfrak{M} \models \varphi[s]$): Given an assignment function s, after we evaluate it and interpret all the relation symbols as well as $\gamma, \vee$ and $\forall$ semantically, is the resulting value true. If two assignment functions that agree on the free variables of a formula φ, then $\mathfrak{M} \models \varphi[s_1]$ if and only if $\mathfrak{M} \models \varphi[s_1]$. Consequently, if φ is a sentence $\mathfrak{M} \models \varphi[s]$ for any variable assignment function s, and so it is meaningful to drop the $[s]$ and just say $\mathfrak{M} \models \varphi$ if $\mathfrak{M} \models \varphi[s]$ for all s.

9. **(Models, satisfiable, truth)**: if $\mathfrak{M} \models \varphi$, w say that $\mathfrak{M}$ models φ. If φ is a formula in $\mathfrak{M}$, we will say that it's *satisfiable* if there is a $\mathfrak{M}$ such that $\mathfrak{M} \models \varphi$. If φ is a sentence, then then if $\mathfrak{M} \models \varphi$, we shall say that φ is *true*.

10. **Existential Quantifier and Shorthand's** Let φ be a formula. Then $\mathfrak{M} \models (\exists x,\ \varphi)[s]$ if and only if there is some $m \in |\mathfrak{M}|$ such that $\mathfrak{M} \models \varphi[s(x|m)]$. Furthermore, for sentences, all the usual rules of logic works for modelling, for example $\mathfrak{M} \models \neg\alpha$ if and only if $\mathfrak{M} \not\models \alpha$. If we were dealing with formulas, we would have to keep track of an assignment function. For formulas in general, we also have:

$$\Sigma \cup \{\psi\} \models \rho \qquad \text{if and only if} \qquad \Sigma \models (\psi \to \rho)$$

11. **Substitution and Substitutability**: replace a particular variable x with a term t. As long as the replacement does not change the truth value, then the substitution is substitutable.

12. **Logical Implications** ($\Delta \models \Gamma$) Let Δ, Γ be two sets of $\mathcal{L}$-formula. Then Δ logically implies Γ if for any structure where $\mathfrak{M} \models \Delta$ then $\mathfrak{M} \models \Gamma$.

13. **Tautology and Contradiction**: If $\emptyset \models \varphi$, then φ is a tautology. If $\emptyset \models \neg\varphi$,then φ is a *contradiction*.

14. **Modelling Contradiction**: If φ is a contradiction, $\mathfrak{M} \models \varphi[s]$ is false for every structure $\mathfrak{M}$ and assignment s. Hence, a set Σ is satisfiable if and only if there is no contradiction χ such that $\Sigma \models \chi$.

15. **Propositional formulas**: no functions, relations, constants. Just variables, connectives, parenthesis, and recursive relations.

16. **Truth Assignment et al**: assign T, F to every variable. Then recursively go through formulas and deal with $\neg$ and $\to$. All the same shorthand's apply for $\vee, \leftrightarrow, \wedge$, and so forth. Same notion of satisfiable, tautology, and contradiction, and logical implication (which we'll call *propositional implication* if the context requires specificity)

2

Deductions

We have now firmly established the language of logic, with the notion of a language having nouns and statements, structures in which we may wonder if those formulas are satisfiable, and we can ask whether a formula or sentences are valid, that is model any structure with any variable-assignment function. We know learn how to string together deductions that will let us step-by-step deduce more complicated results by incrementally showing each step of the way a true statement that let's us deduce the next. In the beginning, we will start by defining the language of deductions (just like how language have yet to have any truth values). In general, a deductive system need not be *sound*, meaning that a deduction will be equivalent to a *logical implication* (i.e. if we deduce B from A, $A \vdash B$, then $A \models B$). This will be called the *soundness theorem*. We will see that the deductive system we will create is indeed sound: that deduction imply implications. In the next chapter, will show the converse, that implication gives deduction: this will be called the *completeness theorem*.

One of the key properties of proof's is that they are (relatively) easy to follow, as compared to the usual difficulty there is in producing them. We would want confidence in being able to know that at the end of a proof, we know it's true and was easily checkable (if at least easy mechanically) to see that it's true. Hence, we be following these restrictions on our logical axioms and rules of inference:

1. There will be an algorithm (i.e., a mechanical procedure) that will decide, given a formula θ, whether or not θ is a logical axiom.
2. There will be an algorithm that will decide, given a finite set of formulas Γ and a formula θ, whether or not (Γ, θ) is a rule of inference.
3. For each rule of inference (Γ, θ), γ will be a finite set of formulas.
4. Each logical axiom will be valid.
5. Our rules of inference will preserve truth. In other words, for each rule of inference (Γ, θ), $\Gamma \models \theta$.

2.1 Deductions

Fix a language $\mathcal{L}$. A deduction will be a finite sequence of formula's which we will consider to be legal steps to take letting us go from our assumptions (if any) to our conclusion. If each step of the sequence follows the rule of deduction, we will get a valid deduction. The legal actions we can take may come from one of three sources:

1. the *logical axioms*: these are a collection of formulas which are part of our deductive system that we may use in any deduction (note that the validity of them is not in question yet, they are simply syntactical formulas)
2. the *nonlogical axioms*: These are a collection of formulas which are given *for a given deduction.* They are the assumptions that are made for a particular deduction
3. the *rules of inference*: These are a collection of rules that will tell us when we are allowed to combine two pieces of information together to get the next step in our deduction.

For now, assume we have a fixed set of $\mathcal{L}$-formula Λ called the *logical axioms*, and set of ordered pairs (Γ, φ) called the *rules of inferences.* In the following sections, we will specify the formulas that belong to the logical axioms and the rules of inferences. The logical axioms are will be formulas that will be deemed to be "justified" in any deduction. Sometimes, we require more context, and so we will sometimes need a context-dependent set Σ to have the justification (thus, a deduction will be dependent on Σ). The rules of inferences will be a way of deducing the next step in a deduction given what was already deduced.

Definition 2.1.1: Deduction

Let Σ be a collection of $\mathcal{L}$-formulas and D a finite sequence of $\mathcal{L}$-formulas $(\varphi_1, \varphi_2, ..., \varphi_n)$. Then we will say that D is a *deduction* from Σ if for each i, $1 \leq i \leq n$, either

1. $\varphi_i \in \Lambda$ (that is φ_i is a logical axiom)
2. $\varphi_i \in \Sigma$ (that is φ_i is a nonlogical axiom)
3. there is a rule of inference (Γ, φ_i) such that $\Gamma \subseteq \{\varphi_1, \varphi_2, ..., \varphi_{i-1}\}$

I there is a deduction from Σ, the last line of which is the formula φ, we will call this a *deduction form* Σ *of* φ and write $\Sigma \vdash \varphi$

Depending on Λ and the rules of inferences, there may be many different types of deductive systems. Often, there is either many logical axioms with few rules of inferences, or many rules of inference with few logical axioms, though this is a rule of thumb.

Example 2.1: Deductions

1. Let $\mathcal{L} = \{P\}$ where P is a binary relation symbol. Let Σ be our set of axioms

$$\Sigma = \{\forall x P(x, x), P(u, v), P(u, v) \to P(v, u), P(v, u) \to P(u, u)\}$$

and let $\Lambda = \emptyset$ for now. For the rules of inferences, let it be

$$\{(\{\alpha, \alpha \to \beta\}, \beta) \mid \alpha \text{ and } \beta \text{ are formulas of } \mathcal{L}\}$$

which is the rule of modus ponens, that is the given the formulas α and $\alpha \to \beta$, we may conclude β (we will abbreviate this to **MP**). Now we can write a deduction from Σ of the formula $P(u,u)$, $\Sigma \vdash P(u,u)$ as follows

$$\begin{aligned} &P(u,v) && \in \Sigma \\ &P(u,v) \to P(v,u) && \in \Sigma \\ &P(v,u) && \textbf{MP},\ (1,2) \\ &P(v,u) \to P(u,u) && \in \Sigma \\ &P(u,u) && \textbf{MP},\ (3,4) \end{aligned}$$

and so, we have deduced $P(u,u)$. Note that we could not have deduce $\forall x P(x,x) \vdash P(u,u)$ since the symbol $\forall$ is still a formality here! We are very restricted in what symbols means, deductions at this stage are still linguistic manipulation rules.

2. Let $\Lambda = \Sigma = \emptyset$. Then given any language $\mathcal{L}$, is it possible to deduce something? In fact, it is not[a]! This captures intuitively the idea that we cannot make deductions without any assumptions.

[a]Once we get to rules of inference, if we allow any tautology for any formula φ once converted into propositional logic φ_p to be in a rule of inference ,then we can actually deduce more. This will be explored when we talk about rules of inference

Proposition 2.1.1: Universal set of Deduction for Σ

Let $\mathcal{L}$ be a language, fix a set of $\mathcal{L}$-formulas Σ and Λ, and a collection of rule inferences. Then the set $Thm_\Sigma \{\varphi \mid \Sigma \vdash \varphi\}$ is the smallest set such that

1. $\Sigma \subseteq Thm_\Sigma$
2. $\Lambda \subseteq Thm_\Sigma$
3. If (Γ, θ) is a rule of inference and $\Gamma \subseteq Thm_\Sigma$ then $\theta \in C$.

***Proof* :**
This is all straight forward elementary 1st year math. See "A friendly introduction to mathematical logic, p. 46" if you need any help.

Example 2.2: Theorem
Take the sentence $\varphi :\equiv \exists x \exists y((\neg x = y) \wedge \forall z(z = x \vee z = y))$. Every $\mathcal{L}$ structure satisfying this sentence must have exactly two elements in the universe. Hence $\{\varphi\}$ is a set of non-logical axioms for the collection of sets of cardinality two:

$$\{\mathfrak{M} \mid \mathfrak{M} \text{ is an } \mathcal{L}\text{-structure with exactly two elements}\}$$

2.2 The Logical Axioms

We know figure out what our logical axioms will be. There will be an infinite amount of them, but we can they will be decidable (a computer in a finite amount of time can decide whether a given formula φ is a logical axiom or not). These will be the axioms we will use for all our structures. For a particular structure, we will amend some more axioms via Σ. We will talk about those and rules of inference later, concentrating for now on logical axioms

2.2.1 Equality Axioms

In these notes, we have taken the approach for $=$ being part of any language. We will have three equality axioms. These axioms (in particular the last 2) are designed to allow substitution of equal for equals

1. The first simply says that any object is equal to itself

$$x = x \text{ for each variable } x \tag{2.1}$$

2. Assume that $x_1, x_2, ..., x_n$ are variable and $y_1, y_2, ..., y_n$ are variables and f is an n-ary function symbol. Then

$$[(x_1 = y_1) \wedge (x_2 = y_2) \wedge \cdots \wedge (x_n = y_n)] \to (f(x_1, x_2, ..., x_n) = f(y_1, y_2, ..., y_n)) \tag{2.2}$$

3. Similarly to the above except replace the function f with the relation R

$$[(x_1 = y_1) \wedge (x_2 = y_2) \wedge \cdots \wedge (x_n = y_n)] \to (R(x_1, x_2, ..., x_n) \to R(y_1, y_2, ..., y_n)) \tag{2.3}$$

   Sometimes, $=$ is seen as a part of this part

2.2.2 Quantifier Axioms

The Quantifier axioms allow for easy substitution. In other words, if we have $\forall x P(x)$, and t is a term in our language, we would like to be able to state $P(t)$, or if we have formula where we can substitute t by x φ_t^x, then we want to say $\exists x \varphi$. Hence the following axioms:

1. (universal substantiation)

$$(\forall x \varphi) \to \varphi_t^x, \text{ if } t \text{ is substitutable for } x \text{ in } \varphi \tag{2.4}$$

2. (existential generalization)

$$\varphi_t^x \to (\exists x \varphi), \text{ if } t \text{ is substitutable for } x \text{ in } \varphi \tag{2.5}$$

2.3 Rules of Inference

We will have two rules of inferences: one for propositional consequences and one for quantifiers. To understand rules of inference, we will briefly dive into *propositional logic*. We will be working

with a restricted language $\mathcal{P}$ consisting of countably infinite propositional variables $A, B, C, ...$ and connectives $\vee$ and $\neg$ (or $\neg$ and $\rightarrow$). We have no quantifiers, no relation symbols, no function symbols, and no constants. Formulas are defined as being the collection of all φ such that φ is a propositional variables, or φ is $(\neg\alpha)$ or φ is $(\alpha \vee \beta)$, with α and β being formulas of propositional logic. Each propositional variable can be assigned one of two truth values: T or F via a truth assignment function v. We then extend v to any formula as so:

$$\overline{v}(\varphi) = \begin{cases} v(\varphi) & \text{if } \varphi \text{ is a propositional variable} \\ \text{F} & \text{if } \varphi :\equiv (\neg\alpha) \text{ and } \overline{v}(\alpha) = T \\ \text{F} & \text{if } \varphi :\equiv (\alpha \vee \beta) \text{ and } \overline{v}(\alpha) = \overline{v}(\beta) = F \\ T & \text{otherwise} \end{cases}$$

With this, we have the following definition:

Definition 2.3.1: Tautology And Contradiction

Let φ be a propositional formula. Then φ Is a *tautology* if $\overline{v}(\varphi) = T$ for any truth assignment v, and a *contradiction* if $\overline{v}(\varphi) = F$ for any truth assignment v.

Propositional logic will be a useful simplification of first-order logic. We will show how to convert a $\mathcal{L}$-formula β into a (propositional) formula β_P

1. Find all subformulas of $|beta$ of the form $\forall x\alpha$ that are not in the scope of another quantifier. Replace them with propositional variables in a systematic fashion. This means that if $\forall y Q(y, c)$ appears twice in β, it is replaced by the same letter both times, and distinct subformulas are replaced by distinct letters
2. Find all atomic formulas that remain, and replace them systematically with new propositional variables

For example, take

$$(\forall x P(x) \wedge Q(c, z)) \rightarrow (Q(c, z) \vee \forall x P(x))$$

Then this would become

$$(B \wedge A) \rightarrow (A \vee B)$$

Note that if β_P is a tautology, then β is valid, but the converse is false

Example 2.3: Valid First Order Formula Not Tautology

Let

$$\beta :\equiv [(\forall x)(\theta) \wedge (\forall x)(\theta \rightarrow \rho)] \rightarrow (\forall x)(\rho)$$

Then β is valid, but β_P would be $[A \wedge B] \rightarrow C$, which his certainly not a tautology.

Definition 2.3.2: Propositional Consequence for Propositional Logic

Let Γ_P be a set of propositional formulas and phi_P a propositional formula. The new will say that φ_P is a propositional consequence of $\Gamma - P$ if every truth assignment that makes each propositional formula in Γ_P true also makes φ_P true. Notice that φ_P is a tautology if and only if φ_P is a propositional consequence of $\emptyset$

Lemma 2.3.1: propositional Consequence Equivalent Formulation

If $\Gamma_P = \{(\gamma_1)_P, ..., (\gamma_n)_P\}$ is a nonempty finite set of propositional formulas and φ_P is a propositional formula, then φ_P is a propositional consequence of Γ_P if and only if

$$[(\gamma_1)_P \wedge (\gamma_2)_P \wedge \cdots \wedge (\gamma_n)_P] \to \varphi_P$$

is a tautology

Proof **:**
easy exercise

Definition 2.3.3: Propositional Consequence For 1st Order Logic

Let Γ be a finite set of $\mathcal{L}$-formulas and φ is an $\mathcal{L}$-formula. Then we will say that φ Is a propositional consequence of Γ if φ_P is a propositional consequence of Γ_P.

Example 2.4: Propositional Consequence For 1st Order Logic
Let $\mathcal{L}$ be a language that contains two relations symbols P and Q. Let Γ be the set

$$\{\forall x P(x) \to \exists y Q(y), \exists y Q(y) \to P(x), \neg P(x) \leftrightarrow (y = z)\}$$

If we let φ be the formula $\forall x P(x) \to \neg(y = z)$, then by applying the conversion procedure we get that

$$\Gamma_P = \{A \to B, B \to C, \neg C \leftrightarrow D\}$$

and

$$\varphi_P :\equiv A \to \neg D$$

Then we can easily see that φ is a propositional consequence of Γ.

Definition 2.3.4: Rule Of Inference Of Type (PC)

Let Γ be a finite set of $\mathcal{L}$-formulas, φ is an $\mathcal{L}$-formula, and φ is a propositional consequence of Γ, then (Γ, φ) is a *rule of inference of type (PC).*

The point behind the formalism is that we managed to prove, say γ_1, γ_2, and we proved that

$$[\gamma_1 \wedge \gamma_2 \to \varphi]_P$$

is a tautology, then we may conclude φ. Furthermore, if φ_P is a tautology, rule (PC) allows us to assert φ in any deduction by setting $\Gamma = \emptyset$.

2.3.1 Quantifier Rules

The idea behind Quantifier rules is simples: If we managed to prove "x is this property" without quantifying $x<$ then it is reasonable to say that we have proved "$(\forall x)x$ is this property". Similarly, if

we proved something with the assumption of "x did this thing", then it is safe to say that we assume "$(\exists x)x$ did this thing", then we can still prove the same thing

Definition 2.3.5: Rules Of Inference Of Type (QR)

Suppose that the variable x Is not free in the formula $\psi>$ Then both of the following are *rules of inference of type (QR)*:

$$(\{\psi \to \varphi\}, \psi \to (\forall x \varphi(x)))$$
$$(\{\varphi \to \psi\}, \psi \to (\exists x \varphi) \to \psi)$$

What we are essentially saying is

1. From the formula $\psi \to \varphi$, we may deduce $\psi \to (\forall x \varphi)$
2. From the formula $\varphi \to \psi$, we may deduce $(\exists x \varphi) \to \psi$

2.4 Soundness

We now make sure that the axioms we have a sound, that is they are all valid

Theorem 2.4.1: Soundness Theorem For Logical Axioms

The logical axioms are valid

Proof **:**
This requires checking the equality axioms and the quantifier axioms. P. 56 of a friendly introduction to Logic

Theorem 2.4.2: Soundness Of Rules Of Inference

Suppose that (Γ, θ) is a rule of inference. Then $\Gamma \models \theta$.

Proof **:**
p. 57 of a friendly introduction to Logic

Theorem 2.4.3: Soundness Theorem

If $\Sigma \vdash \varphi$ then $\Sigma \models \varphi$

Proof **:**
p. 57

(two technical lemmas are required for the above theorem, they are presented on p.58 onwards in the book)

2.5 Examples

here

2.6 nonlogical axioms

here

2.7 Completeness and Compactness

here

2.8 *Deduction's with Propositional Logic

We take a moment to develop a different deductive system that is still sound. This deductive system will be for propositional logic. For any language $\mathcal{L}$, we will label $\mathcal{L}_p$ the propositional language induced by $\mathcal{L}$. in this section, the term formula will refer to *propositional formula*.

Definition 2.8.1: Axiom Schema

For any formulas' $\alpha, \beta, \gamma \in \mathcal{L}_P$, the following are valid axioms:

A1 $(\alpha \to (\beta \to \alpha))$

A2 $((\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)))$

A3 $(((\neg\beta) \to (\neg\alpha)) \to (((\neg\beta) \to \alpha) \to \beta))$

These three are called the *axiom schemas*, and any particular choice of formulas gives an *axiom* of $\mathcal{L}_P$

this is the equivalent to the logical axioms Λ. It should be shown that every axiom of $\mathcal{L}_P$ is a tautology (and hence, if converted from syntactic to semantic meaning, it is already sound).

The rule of inference we will have is simply Modus Ponens

Definition 2.8.2: Modus Ponens (MP)

Given any formula φ and $(\varphi \to \psi)$, then we may infer ψ, i.e.

$$(\{\varphi, (\varphi \to \psi)\}, \psi)$$

is our rule of inference.

It is important for future reference to box the following result:

Proposition 2.8.1: MP and axiom Satisfies Logical Implication

The logical axioms are tautologies, and

$$\{\varphi, (\varphi \to \psi)\} \models \psi$$

Don't forget that as of yet, MP is just a rule by which we can add further formulas to our deduction and doesn't have a semantic meaning. If you wish, you can think that it simply happens that MP satisfies the above formula

***Proof* :**
A quick truth table shows this result

The definition of deduction remains the same as for first-order logic.

Example 2.5: Deductions In Propositional Logic

1. We'll show $\vdash \varphi \to \varphi$

$(\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$	A2
$\varphi \to ((\varphi \to \varphi) \to \varphi)$	A1
$\varphi \to ((\varphi \to \varphi) \to \varphi) \to (\varphi \to \varphi)$	1, 2 MP
$\varphi \to (\varphi \to \varphi)$	A1
$\varphi \to \varphi$	3,4 MP

2. We'll next show transitivity: $\{\alpha \to \beta, \beta \to \gamma\} \vdash \alpha \to \gamma$:

$(\beta \to \gamma) \to (\alpha \to (\beta \to \gamma))$	A1
$\beta \to \gamma$	Premise
$\alpha \to (\beta \to \gamma)$	1,2 MP
$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	A2
$(\alpha \to \beta) \to (\alpha \to \gamma)$	4,3 MP
$\alpha \to \beta$	Premise
$\alpha \to \gamma$	5,6 MP

Hence, $\{\alpha \to \beta, \beta \to \gamma\} \vdash \alpha \to \gamma$, completing the proof.

3. Since we've shown the above two deductions, which is simply a finite length of sequences, we may always simply add in the deduction of them into any other deduction. For example, we'll show $\vdash (\neg\alpha \to \alpha) \to \alpha$

$(\neg\alpha \to \neg\alpha) \to ((\neg\alpha \to \alpha) \to \alpha)$	A3
$(\neg\alpha \to \neg\alpha)$	first example
$(\neg\alpha \to \alpha) \to \alpha$	1,2 MP

   completing the proof.

4. Another useful one to have is $\{\alpha \to (\beta \to \gamma), \beta\} \vdash \alpha \to \gamma$. See if you can figure this out (commented in LaTeXdocument)

5. For $\vdash \alpha \vee \neg\alpha$, remember to first convert to $\neg a \to b$, which in this case is $(\neg\alpha) \to \neg\alpha$, which we've shown is deducible in example 1.

6. A result that combines many of the above results is the following: $\vdash \neg\neg\beta \to \beta$

$(\neg\beta \to \neg\neg\beta) \to ((\neg\beta \to \neg\beta) \to \beta)$	A3
$(\neg\neg\beta \to (\neg\beta \to \neg\neg\beta))$	A1
$\neg\neg\beta \to ((\neg\beta \to \neg\beta) \to \beta)$	1,2 and example 2
$\neg\beta \to \neg\beta$	example 1
$\neg\neg\beta \to \beta$	3,4 example 4

   completing the proof.

There are many important shortcuts that one ought to know

Lemma 2.8.1: Deduction Shortcuts

1. If $(\varphi_1, \varphi_2, ..., \varphi_n)$ is a deduction n of $\mathcal{L}_P$, then $\varphi_1, \varphi_2, ..., \varphi_l$ is a deduction of $\mathcal{L}_P$, $1 \leq l, \leq n$
2. If $\Gamma \vdash \delta$, $\Gamma \vdash \delta \to \beta$, then $\Gamma \vdash \beta$
3. If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \alpha$, then $\Delta \vdash \alpha$
4. If $\Gamma \vdash \Delta$, $\Delta \vdash \sigma$, then $\Gamma \vdash \sigma$

***Proof* :**
all very easy proofs.

An important result is that two opposite formulas can deduce anything:

Theorem 2.8.1: Contradiction Deduces Anything

For any δ and γ

$$\{\delta, \neg\delta\} \vdash \gamma$$

showing that having (intuitively) a premise and the negation of the premise can imply anything

***Proof* :**

(1)	$\neg\delta \to (\neg\gamma \to \neg\delta)$	A1
(2)	$\neg\delta$	Premise
(3)	$\neg\gamma \to \neg\delta$	1,2 MP
(4)	$\delta \to (\neg\gamma \to \delta)$	A1
(5)	δ	Premise
(6)	$\neg\gamma \to \delta$	4,5 MP
(7)	$(\neg\gamma \to \neg\delta) \to ((\neg\gamma \to \delta) \to \gamma)$	A3
(8)	$(\neg\gamma \to \delta) \to \gamma$	3,7 MP
(9)	γ	6, 8 MP

One important simplification that is important but more complicated to prove is the following

Theorem 2.8.2: Deduction Theorem

If Σ is any set of formulas, α and Γ are any formula, then $\Sigma \vdash \alpha \to \beta$ if and only if $\Sigma \cup \{\alpha\} \vdash \beta$

***Proof* :**

The fact that $\Sigma \vdash (\alpha \to \beta)$, then $\Sigma \cup \{\alpha\} \vdash \beta$ is easy: simply use Modus Ponens.

For the other direction, assume $\Sigma \cup \{\alpha\} \vdash \beta$. We will use induction on the minimal length of deduction (which exists since we are dealing with discrete quantities bounded bellow by a minimal length of 1). If $n = 1$, then it must be that $\beta \in \Sigma$, and so using axiom 1 we have:

β	$\beta \in \Sigma$
$\beta \to (\alpha \to \beta)$	A1
$\alpha \to \beta$	1,2 MP

So let's say it holds true for $k < n$ so that for any δ, γ if a shortest deduction $\Sigma \cup \{\delta\} \vdash \gamma$ has length k then $\Sigma \vdash (\delta \to \gamma)$. Since the length of the deduction is longer than 1, it cannot be that $\beta \in \Sigma$ (similarly, it cannot be a logical axiom, or else it would a minimal length of 1). Let's take a deduction $\Sigma \cup \{\alpha\} \vdash \beta$ with a shortest length deduction n. Since we conclude β, it must be we use modus ponens to arrive at β in the deduction $\Sigma \cup \{\alpha\} \vdash \beta$, meaning we have φ_i, φ_j with $i, j < n$ with which we use MP. Note that $\max(i, j) = n - 1$, since if it were less, then we would a shortest deduction of length less than n, contradicting our assumption of n being the minimal length. Thus, let

$$\varphi_i = \theta \qquad \varphi_j = \theta \to \beta$$

We now have two deduction, each of length less than n:

$$\Sigma \cup \{\alpha\} \vdash \theta \qquad \Sigma \cup \{\alpha\} \vdash (\theta \to \beta)$$

hence, by our induction hypothesis, we also have:

$$\Sigma \vdash \alpha \to \theta \qquad \Sigma \vdash \alpha \to (\theta \to \beta) \tag{2.6}$$

We know finish up: concatenate the deduction's in equation (2.6). Then, add to it the following version of axiom 2:

$$[\alpha \to (\theta \to \beta)] \to (\alpha \to \theta) \to (\alpha \to \beta)]$$

Then since we have $\alpha \to (\theta \to \beta)$, by modus ponens we have

$$(\alpha \to \theta) \to (\alpha \to \beta)$$

and since we have $\alpha \to \theta$, by modus ponens again we have:

$$\alpha \to \beta$$

Hence, we have

$$\Sigma \vdash \alpha \to \beta$$

completing the proof.

Using this theorem, we can make some results easier:

Example 2.6: Using Deduction Theorem

1. $\vdash \varphi \to \varphi$ if and only if $\{\varphi\} \to \varphi$, which is trivial.
2. Since $\vdash \neg\neg\beta \to \beta$, it is easy to show $\vdash \beta \to \neg\neg\beta$ in combination with the Deduction theorem, in particular showing that $\beta \vdash \neg\neg\beta$. Letting $\alpha :\equiv (\neg\beta)$, we have

(1) $\neg\neg(\neg\beta) \to \neg\beta$	$\vdash \neg\neg\alpha \to \alpha$
(2) $(\neg\neg(\neg\beta) \to \neg\beta) \to ((\neg\neg\neg\beta \to \beta) \to \neg\neg\beta)$	A3
(3) $(\neg\neg\neg\beta \to \beta) \to \neg\neg\beta$	1,2 MP
(4) $\beta \to (\neg\neg\neg\beta \to \beta)$	A1
(5) β	Premise
(6) $(\neg\neg\neg\beta \to \beta)$	4,5 MP
(7) $\neg\neg\beta$	3,6 MP

3. $\neg\neg a \to b \vdash a \to b$. By the deduction theorem, we have $\{\neg\neg a \to b, a\} \vdash b$. Since $a \vdash \neg\neg a$, we have

$\neg\neg a \to b$	premise
$\neg\neg a$	$a \vdash \neg\neg a$
b	

   completing the proof. Similarly, since $\neg\neg a \vdash a$, we can prove $a \to b \vdash \neg\neg a \to b$

Another important result is Modus tollens

Lemma 2.8.2: Modus Tollens (MT)

$$\neg\beta \to \neg\alpha \vdash \alpha \to \beta \qquad \text{and} \qquad \alpha \to \beta \vdash \neg\beta \to \neg\alpha$$

***Proof* :**
To deduce he first, by the deduction theorem:

$$\{\neg\beta \to \neg\alpha, \alpha\} \to \beta$$

Then

(1) $(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$	A3
(2) $(\neg\beta \to \neg\alpha)$	Premise
(3) $((\neg\beta \to \alpha) \to \beta)$	1,2 MP
(4) $\alpha \to (\neg\beta \to \alpha)$	A1
(5) α	Premise
(6) $\neg\beta \to \alpha$	4,5 MP
(7) β	3,6 MP

For the second, by the Deduction theorem it is equivalent to show that

$$\{\alpha \to \beta, \neg\beta\} \vdash \neg\alpha$$

Hence, it suffice to deduce $\neg\alpha$. We proceed with the deduction:

(1)	$\neg\beta \to (\alpha \to \neg\beta)$	A1
(2)	$\neg\beta$	Premise
(3)	$\alpha \to \neg\beta$	1,2 MP
(4)	$(\neg\neg\alpha \to \neg\beta)$	3, deduced in example
(5)	$(\neg\neg\alpha \to \neg\beta) \to ((\neg\neg\alpha \to \beta) \to \neg\alpha)$	A3
(6)	$(\neg\neg\alpha \to \beta) \to \neg\alpha$	4,5 MP
(7)	$(\alpha \to \beta)$	Premise
(8)	$(\neg\neg\alpha \to \beta)$	7, deduced in example
(9)	$\neg\alpha$	6,8 MP

Showing that $\{\alpha \to \beta, \neg\beta\} \to \neg\alpha$, which by the Deduction theorem is equivalent to

$$\alpha \to \beta \vdash \neg\beta \to \neg\alpha$$

completing the proof.

Corollary 2.8.1: Consequences Of MT

1. $\alpha \vdash \beta$ if and only if $\neg\beta \vdash \neg\alpha$
2. $\{\alpha \wedge \beta\} \vdash \neg\beta \vdash \neg\alpha$
3. $(\beta \to \neg\alpha) \vdash (\alpha \to \neg\beta)$
4. $(\neg\beta \to \alpha) - \vdash (\neg\alpha \to \beta)$
5. $\sigma \vdash (\sigma \vee \tau)$
6. $\{\alpha \wedge \beta\} \vdash \beta$
7. $\{\alpha \wedge \beta\} \vdash \alpha$

***Proof* :**

1. By the deduction theorem, $a \vdash b$ if and only if $\vdash a \to b$. By MT $a \to b \vdash \neg b \to \neg a$. Thus, by transitivity, we have $\vdash \neg b \to \neg a$. By the deduction theorem $\vdash \neg b \to \neg a$ if and only if $\neg b \vdash \neg a$, showing one side of the implication. For the other side, we use the other side of MT to get $\neg b \to \neg a \vdash a \to b$. Hence:

$$a \vdash b \;\Leftrightarrow \vdash a \to b \;\Leftrightarrow \vdash \neg b \to \neg a \;\Leftrightarrow\; \neg b \vdash \neg a$$

   Giving us that $a \vdash b$ if and only if $\neg b \vdash \neg a$, completing the lemma.

2. Recall that $\alpha \wedge \beta :\equiv \neg(\alpha \to (\neg\beta))$. This is equivalent to showing $\neg(\alpha \to (\neg\beta)) \vdash \neg(\beta \to (\neg\alpha))$. Then:

$$(\beta \to (\neg\alpha)) \vdash (\alpha \to (\neg\beta))$$

   By MT:

$$\neg\neg\alpha \to \neg\beta$$

   We showed earlier that: $\neg\neg a \to b \vdash a \to b$, and so we get to deduce

$$\alpha \to \neg\beta$$

   as we sought to show.

The rest are left as exercises.

2.8.1 Soundness and Completeness For Propositional Logic

With that established, we may ask about the soundness of this deductive system, in particular if Δ is a set of formulas and α is a formula such that $\Delta \vdash \alpha$, then $\Delta \models \alpha$. The soundness theorem is relatively easy to prove

Theorem 2.8.3: Soundness Theorem

Let Δ be a set of formulas and α a formula such that $\Delta \vdash \alpha$. Then $\Delta \models \alpha$

Proof **:**
We will proceed by induction on the shortest length deduction from Δ to α. If $n = 1$, then that either means α is an axiom or $\alpha \in \Delta$. If α is an axiom, which is easy to show they are all tautologies (hence satisfying all truth assignment). Thus, if $v(\Delta) = T$, then $v(\alpha) = T$, and so $\Delta \models \alpha$

Assume now that $\Delta \vdash \beta$ then $\Delta \models \beta$ for any deduction of length $k < n$, and consider $\Delta \vdash \alpha$ has shortest deduction of length $n > 1$. Then it must be that we conclude α by Modus Ponens, since if it were by an axiom or if $\alpha \in \Delta$, the deduction would be of length 1. We thus must have

$$\varphi_i :\equiv \gamma \qquad \varphi_j :\equiv \gamma \to \alpha \qquad \text{for some } i, j < n$$

Then by modus ponens we'd get $\varphi_n :\equiv \alpha$. Now, since $\Delta \vdash \gamma$ and $\Delta \vdash \gamma \to \alpha$, by our induction hypothesis we know that $\Delta \models \gamma$ and $\Delta \models \gamma \to \alpha$. Since:

$$v(\gamma \to \alpha) = \begin{cases} F & \text{if } v(\gamma) = T \text{ or } v(\alpha) = F \\ T & \text{otherwise} \end{cases}$$

and $v(\gamma) = v(\gamma \to \alpha) = T$, it must be that $v(\alpha) = T$, implying $\Delta \models \alpha$. Thus, by the principle of induction, $\Delta \vdash \alpha$ then $\Delta \models \alpha$, completing the proof.

We start working towards the completeness theorem. We require some more definitions and lemma

Definition 2.8.3: Inconsistent

Let Γ be a set of formulas. Then Γ is *inconsistent* if $\Gamma \vdash \neg(\alpha \to \alpha)$ for some formula α. Γ is *consistent* if it is not inconsistent

The main result we'll show is that satisfiability and consistency are "equivalent" for propositional logic which will lead us to the completeness theorem:

Lemma 2.8.3: Satisfiable, then Consistent

Let Σ be a set of satisfiable formulas (so that there is a v such that $v(\Sigma) = T$). Then Σ is consistent (so $\Sigma \nvdash \neg(\alpha \to \alpha)$)

Proof **:**
Assume Σ is satisfiable with the truth-assignment function v. For the sake of contradiction, let's say Σ is inconsistent, meaning $\Sigma \vdash \neg(\alpha \to \alpha)$ with a deduction of length n. Recall that the logical axioms are tautologies ($\models \alpha$ for any axiom), and by proposition 2.8.1 ($\{\alpha, (\alpha \to \beta)\} \models \beta$). Hence, whether φ_i is part of the axiom schema, a result of modes ponens, or an element of Σ, we have

that

$$\begin{aligned} \Sigma &\models \varphi_1 \\ \Sigma \cup \{\varphi_1\} &\models \varphi_2 \\ &\vdots \\ \Sigma \cup \{\varphi_i \mid 1 \leq i \leq n-1\} &\models \varphi_n \end{aligned}$$

since $\varphi_n :\equiv \neg(\alpha \to \alpha)$, that means we have implied a contradiction. However, the set on the left hand side is valid, meaning the right hand side would have to be true. Since it's a contradiction, we must have:

$$\Sigma \cup \{\varphi_i \mid 1 \leq i \leq n-1\} \not\models \varphi_n$$

However, a valid set cannot be imply and not imply a statement, hence, it cannot be that Σ is inconsistent, completing the proof.

For the converse, we would require to show that if Σ is consistent, we can find a truth assignment that satisfies it. We will build up to exactly this result, which will be they key to the soundness theorem. We first explore some property of inconsistent and consistent sets:

Lemma 2.8.4: Inconsistent Can Deduce Anything

Suppose that Δ is an inconsistent set of formulas. Then $\Delta \vdash \psi$ for any formula ψ

***Proof* :**
Since Δ is inconsistent, we have $\Delta \vdash \neg(\alpha \to \alpha)$. Let ψ be any formula. Then as shown in example 2.5 $\vdash (\alpha \to \alpha)$, and so by the elementary lemma's we have $\neg\psi \vdash (\alpha \to \alpha)$, and so by the deduction theorem we have $\vdash \neg\psi \to (\alpha \to \alpha)$. By Modus Tollens, we have $\vdash \neg(\alpha \to \alpha) \to \neg\neg\psi$. Combining with Δ, we get that $\Delta \vdash \neg(\alpha \to \alpha) \to \neg\neg\psi$. Since $\Delta \vdash \neg(\alpha \to \alpha)$ we have by modus tollens $\Delta \vdash \neg\neg\varphi$. Finally, since $\neg\neg\varphi \vdash \varphi$, by transitivity of deductions we have:

$$\Delta \vdash \psi$$

completing the proof.

Proposition 2.8.2: Inconsistent, Then Only Requires Finitely Many Formulas

Suppose Σ is an inconsistent set of formulas. Then there is a finite subset Δ of Σ such that Δ is inconsistent

***Proof* :**
Suppose that Σ is inconsistent so that $\Sigma \vdash \neg(\alpha \to \alpha)$. Then by definition of a deduction a finite set $\Delta \subseteq \Sigma$ is used in the deduction. But then $\Delta \vdash \neg(\alpha \to \alpha)$, completing the proof.

Corollary 2.8.2: Finite Subcondition For Consistency

A set of formulas Γ is consistent if and only if every finite subset of Γ is consistent

Proof **:**
If Γ is inconsistent, then certainly a finite subset is inconsistent by the above. Conversely, suppose one subset $\Delta \subseteq \Gamma$ were inconsistent, meaning $\Delta \vdash \neg(\alpha \to \alpha)$. But then $\Gamma \vdash \neg(\alpha \to \alpha)$, completing the proof.

Finally, we have one more definition needed which will allow us to make consistency imply satisfiability

Definition 2.8.4: Maximally Consistent

A set of formulas Σ is *maximally consistent* if Σ is consistent but $\Sigma \cup \{\varphi\}$ is inconsistent for any $\varphi \notin \Sigma$

Such set certainly exist: if v to be a truth assignment, then $\Sigma = \{\varphi \mid v(\varphi) = T\}$ is maximally consistent: lemma 2.8.3 shows it's consistent, and it is maximal since if $v(\varphi) = F$, then $v(\neg\varphi) = T$ so $\neg\varphi \in \Sigma$. If φ was in Σ as well, we can use lemma 2.8.4 to deduce anything, in particular $\Sigma \cup \{\varphi\} \vdash \neg(\alpha \to \alpha)$.

Proposition 2.8.3: Properties Of Maximally Consistent Set

Suppose that Σ is maximally consistent set of formula and φ and ψ is a formula.

1. If $\Sigma \vdash \varphi$, then $\varphi \in \Sigma$
2. $\neg\varphi \in \Sigma$ if and only if $\varphi \notin \Sigma$
3. $\varphi \to \psi \in \Sigma$ if and only if $\varphi \notin \Sigma$ or $\psi \in \Sigma$

Proof **:**

1. Let's say $\Sigma \vdash \varphi$ so that $(\varphi_1, \varphi_2, ..., \varphi_n, \varphi)$ is a valid deduction. For the sake of contradiction, let's say
$$\Sigma \cup \{\varphi\} \vdash \neg(\alpha \to \alpha)$$
using the deduction theorem, we get:
$$\Sigma \vdash \varphi \to (\neg(\alpha \to \alpha))$$
and then we have that $\Sigma \vdash \varphi$ to use modus ponens to get to the contradiction.
2. Let's say $\varphi \in \Sigma$ and Σ is maximally consistent. Consider $\Sigma \cup \{\neg\varphi\} \vdash \neg(\alpha \to \alpha)$. Since $\varphi \in \Sigma$, we have the subset $\{\varphi, \neg\varphi\}$. But then, we have shown in theorem 2.8.1 that
$$\{\varphi, \neg\varphi\} \vdash \neg(\alpha \to \alpha)$$
Since $\{\varphi, \neg\varphi\} \subseteq \Sigma \cup \{\neg\varphi\}$, we have by lemma 2.8.1(2) that
$$\Sigma \cup \{\neg\varphi\} \vdash \neg(\alpha \to \alpha)$$

showing that $\Sigma \cup \{\neg\varphi\}$ is inconsistent, hence $\neg\varphi \notin \Sigma$.

For the converse, we'll show that if $\Sigma \cup \{\varphi\} \vdash \neg(\alpha \to \alpha)$ and $\Sigma \cup \{\neg\varphi\} \vdash \neg(\alpha \to \alpha)$, then we have a contradiction. Starting with the first one, we get by the deduction theorem that

$$\Sigma \vdash (\varphi \to \neg(\alpha \to \alpha)) \overset{\text{MT}}{\vdash} (\neg\neg(\alpha \to \alpha) \to \varphi) \overset{\text{ex. 2.6(3)}}{\vdash} (\alpha \to \alpha) \to \varphi$$

Hence, by transitivity we have:

$$\Sigma \vdash (\alpha \to \alpha) \to \varphi$$

Since $\vdash \alpha \to \alpha$ by modus ponens and transitivity we get:

$$\Sigma \vdash \varphi$$

which as we've proven earlier implies $\varphi \in \Sigma$. Now, notice that we can do the exact same thing but with $\neg\varphi$ the whole time, leading us to conclude that $\neg\varphi \in \Sigma$. However, as we've seen having φ and $\neg\varphi$ in Σ is a contradiction. Our initial assumption was that neither was inside there, hence it must be that *at least* one is inside there. By the first part of the proof (outlined in the slides and not put as part of the proof for this quiz), we have that *at most* one of them is inside there, completing the proof of the entire proposition.

3. Assume $\varphi \to \psi \in \Sigma$. We want to show that either $\varphi \notin \Sigma$ or $\psi \in \Sigma$ (or both). If $\varphi \notin \Sigma$ we're done, and if $\varphi \in \Sigma$, then by Modus Ponens $\psi \in \Sigma$.

   Conversely, assume either $\varphi \notin \Sigma$ or $\psi \in \Sigma$. BY part 1, it is equivalent to show that either $\neg\varphi \in \Sigma$ or $\psi \in \Sigma$ (which we can intuitively think of the logically equivalent statement for $\varphi \to \psi$ of $\neg\varphi \vee \psi \in \Sigma$). If $\psi \in \Sigma$, then by axiom 1

$\psi \to (\varphi \to \psi)$	A1
ψ	Premise
$\varphi \to \psi$	1,2 MP

   If $\neg\varphi \in \Sigma$, then by a similar logic to what we've just shown $\neg\psi \to \neg\varphi \in \Sigma$. By modus Tollens, $\varphi \to \psi \in \Sigma$.

This proofs are very exciting: if we interpret this in terms of truth assignments and Σ being satisfied, then $\varphi \to \psi \in \Sigma$ if either φ is false or ψ is true, which essentially translates to the inclusion relation just proved above. Similarly, if φ is true, certainly $\neg\varphi$ is not true and hence not in Σ. This allows us to define the following:

Proposition 2.8.4: Truth Assignment Induced By Maximally Consistent Set

Let Σ be a maximally consistent set. Then define a function v by:

$$v(\varphi) = \begin{cases} T & \varphi \in \Sigma \\ F & \text{Otherwise} \end{cases}$$

Then v is a truth assignment.

Proof **:**
a truth assignment function assigns random values to the atomic formulas and then has to recursively satisfy:

$$v(\neg\alpha) = \begin{cases} F & v(\alpha) = T \\ T & v(\alpha) = F \end{cases} \quad v(\alpha \to \beta) = \begin{cases} T & \text{If } v(\alpha) = T \text{ and } v(\beta) = F \\ T & \text{otherwise} \end{cases}$$

But this property is exactly met by the definition of Σ.

Proposition 2.8.5: Maximally Consistent, Then Satisfiable

Let Σ be maximally consistent. Then Σ is satisfiable

Proof **:**
If Σ is a maximally consistent set, then the v in proposition 2.8.4 satisfies it.

Thus, we know have that if Σ is maximally consistent, it is satisfiable. Certainly if $\Delta \subseteq \Sigma$ and Σ is maximally consistent, then Δ is satisfiable. We want to show that any consistent formula is satisfiable. For this, we require the following important result showing us that any consistent set of formula belongs to a maximally consistent set of formulas:

Proposition 2.8.6: Embedding consistent set into Maximally consistent set

Suppose Γ is a consistent set of formulas. Then there is a maximally consistent set of formulas Σ such that $\Gamma \subseteq \Sigma$

Proof **:**
First, notice that by our definition of formula there can only be countably many possible formulas, so let's enumerate them:

$$\{\varphi_0, \varphi_1, ..., \varphi_n, ...\}$$

We will construct such a maximally consistent set Σ inductively by defining $\Gamma = \Sigma_0$ and recursively defining Σ_i to make

$$\Sigma = \cup_{i \in \mathbb{N}} \Sigma_i$$

The base case is trivial ($\Sigma_0 = \Gamma$). For the inductive step, assume that Σ_n is a consistent set of formulas. We have one of two cases: if $\Sigma_n \vdash \varphi_n$, then let $\Sigma_{n+1} = \Sigma_n \cup \{\varphi_n\}$. Otherwise, let $\Sigma_{n+1} = \Sigma_n \cup \{\neg\varphi_n\}$.

First, Σ_{n+1} is consistent. In the first case, if $\Sigma_n \cup \{\varphi_n\}$ were inconsistent, then we can use the deduction theorem and modus ponens to show that Σ_n is inconsistent – a contradiction. In the second case, if $\Sigma_n \cup \{\neg\varphi_n\}$ where inconsistent, then we apply the deduction theorem, then modus tollens, and use the fact that $\vdash (\alpha \to \alpha)$ to get that $\Sigma_n \vdash \varphi_n$ – a contradiction by the construction of Σ_{n+1}. Hence, Σ_{n+1} is consistent.

Finally, take $\Sigma = \bigcup_{i \in \mathbb{N}} \Sigma_i$. This set is consistent since if for the sake of contradiction we have $\Sigma \vdash \neg(\alpha \to \alpha)$, then by definition we have a derivations

$$(\varphi_1, ..., \varphi_{n-1}, \neg(\alpha \to \alpha))$$

If any of the φ_i belong to some Σ_j, then there must be some large enough $k \in \mathbb{N}$ such that all of these formula's belong to Σ_k (by definition of the construction). But then, we have that

$$\Sigma_k \vdash \neg(\alpha \to \alpha)$$

contradicting the fact that Σ_k is consistent. Hence, Σ is consistent. For maximal consistent, consider some $\varphi \notin \Sigma$ and take $\Sigma \cup \{\varphi\}$. Since there are only countably many formula, there is a $i \in \mathbb{N}$ such that $\varphi = \varphi_i$. Then by construction $\neg\varphi \in \Sigma_i \subseteq \Sigma$. But then Σ contains both φ and $\neg\varphi$, which as we've shown means it can imply anything, including $\neg(\alpha \to \alpha)$. Since making Σ bigger makes it inconsistent, we have that Σ is indeed maximally consistent, completing the proof.

Theorem 2.8.4: Consistent If And Only If Satisfiable

A set Σ is consistent if and only if it is satisfiable

Proof **:**
If Σ is satisfiable, by lemma 2.8.3, it is consistent. Conversely, if Σ is consistent, by proposition 2.8.6 it belongs to a maximally consistent set, which by proposition 2.8.4 it is satisfiable, and hence any subset is also satisfiable, completing the proof.

We finally got the Completeness theorem

Theorem 2.8.5: Completeness Theorem

Let Σ be a set of formulas. Then if:

$$\Sigma \models \varphi \qquad \text{then} \qquad \Sigma \vdash \varphi$$

Proof **:**
Say $\Sigma \models \varphi$. If Σ is inconsistent, then it proves a anything, and it is always the case that $\Sigma \vdash \varphi$.

Assuming Σ is consistent, we prove the contrapositive: $\Sigma \nvdash \varphi$ then $\Sigma \not\models \varphi$. In the case where $\neg\varphi \in \Sigma$, Then it cannot be that both $\neg\varphi$ and φ are true, so it cannot be that $\Sigma \models \varphi$, hence $\Sigma \not\models \varphi$. If $\neg\varphi \notin \Sigma$, I claim that $\Sigma \cup \{\neg\varphi\}$ is consistent. To see this, extend Σ to a maximally consistent set $\overline{\Sigma}$ where we know that $\Sigma \nvdash \varphi$ so we make $\Sigma_1 = \Sigma \cup \{\neg\varphi\}$. Then we know that $\overline{\Sigma} \nvdash \varphi$ or else $\varphi \in \overline{\Sigma}$ which we know is not the case. But then $\Sigma \cup \{\neg\varphi\}$ is certainly consistent.

Since $\Sigma \cup \{\neg\varphi\}$ is consistent, it belongs to a maximally consistent set and is thus satisfiable. But if the maximal set is satisfiable, certainly any subset is satisfiable, including the set $\Sigma \cup \{\neg\varphi\}$ But then, there is a truth assignment in which $\Sigma \models \neg\varphi[v]$. But then that means $\Sigma \not\models \varphi[v]$. Thus, $\Sigma \not\models \varphi$.

Thus, we have that if $\Sigma \models \varphi$, then $\Sigma \vdash \varphi$, completing the proof.

One result we get from this that might seem innocuous but is really important is that satisfiability is really a finite condition:

Theorem 2.8.6: Compactness Theorem

A set of formulas Γ is satisfiable if and only if every finite subset of Γ is satisfiable.

Proof **:**
Γ is satisfiable if and only if it is consistent, and a set is consistent if and only if every finite subset is consistent, and a set is consistent if and only if it is satisfiable.

2.9 A second look at Deductions

When we first studied first-order deduction, we have taken a inference heavy approach. In this section, we will take a axiom heavy approach similar to what we've done for propositional deductions. A lot of the work is the same, what will differ is that we will define a large axiom schema and only have modus ponens. Since we are using axioms, we will require the following when it comes to working with quantifiers

Definition 2.9.1: Generalization

If φ is any formula and $x_1, x_2, ..., x_n$ is any variables, then $\forall x_1, ..., \forall x_n \varphi$ is a *generalization* of φ

Example 2.7: Generalizations
Let $\varphi :\equiv x = y \to fx = fy$. Then $\forall z(x = y \to fx = fy)$ and $\forall x \forall y(x = y \to fx = fy)$ are generalizations of φ. $\forall x \exists y \varphi$ is not a generalization.

Lemma 2.9.1: Generalization Of Tautology

Let φ be a tautology (so that $\models \varphi$). Then any generalization of a tautology is a tautology

Proof **:**
Let $\models \varphi$. and consider $\forall x_i \varphi$. Since φ is true for all assignment functions, including $s[x_i|a]$ for all a in any universe, certainly $\models \forall x_i \varphi$ for any variable x_i. This can be continued inductively for all variables.

Definition 2.9.2: First Order Axiom Schema

Every first-order language $\mathcal{L}$ has eight *logical axiom schemas*:

1. **A1** $(\alpha \to (\beta \to \alpha))$
2. **A2** $((\alpha \to (\beta \to \gamma)) \to (\alpha \to \beta) \to (\alpha \to \gamma)))$
3. **A3** $(((\neg\beta) \to (\neg\alpha)) \to ((\neg\beta) \to \alpha) \to \beta))$
4. **A4** $(\forall x\alpha \to \alpha_t^x)$ if t is substitutable for x in α
5. **A5** $(\forall x(\alpha \to \beta) \to (\forall x\alpha \to \forall x\beta))$
6. **A6** $(\alpha \to \forall x\alpha)$ if x does not occur free in α (i.e. if x is bounded or does not exist)
7. **A7** $x = x$
8. **A8** $(x = y \to (\alpha \to \beta))$ if α is atomic and β is obtained by replacing some occurrences (possibly all or none) of x in α by y.

all of these are called the *logical axioms* of $\mathcal{L}$ by replacing for formulas α, β, γ of $\mathcal{L}$. Furthermore, all generalizations of logical axioms of $\mathcal{L}$ are logical axioms of $\mathcal{L}$.

Notice that for **A4**, we may write $\forall x\alpha \to \alpha_x^x$ which is $\forall x\alpha \to \alpha$. All axioms are tautologies:

Proposition 2.9.1: Logical Axiom And Tautology

Every logical axiom is a tautology

Proof **:**
check

This time, instead of having many rules of inference, we'll only have one:

Definition 2.9.3: First Order Modus Ponens

Given a formula φ and $(\varphi \to \psi)$, we may infer ψ.

The definition of deduction is identical

Definition 2.9.4: Deduction Reiteration

Let Σ be a collection of $\mathcal{L}$-formulas and D a finite sequence of $\mathcal{L}$-formulas $(\varphi_1, \varphi_2, ..., \varphi_n)$. Then we will say that D is a *deduction* from Σ if for each i, $1 \leq i \leq n$, either

1. $\varphi_i \in \Lambda$ (that is φ_i is a logical axiom)
2. $\varphi_i \in \Sigma$ (that is φ_i is a nonlogical axiom)
3. there is a rule of inference (Γ, φ_i) such that $\Gamma \subseteq \{\varphi_1, \varphi_2, ..., \varphi_{i-1}\}$

I there is a deduction from Σ, the last line of which is the formula φ, we will call this a *deduction form* Σ *of* φ and write $\Sigma \vdash \varphi$

Example 2.8: First Order Deductions

1. We'll show $\{\alpha\} \vdash \exists x\alpha$

$(\forall x \neg\alpha \to \neg\alpha) \to (\alpha \to \neg\forall x \neg\alpha)$	MT
$\forall x\ \neg\alpha \to \neg\alpha$	A4
$\alpha \to \neg\forall x \neg\alpha$	MP
α	Assumption
$\neg\forall x \neg\alpha$	MP
$\exists x\alpha$	

2. Show that $\vdash \forall x\varphi \to \forall y\varphi^x_y$ if y does not occur at all in φ
3. $\vdash \alpha \vee \neg\alpha$
4. $\{c = d\} \vdash \forall z Qacv \to Qazd$
5. $\vdash x = y \to y = x$
6. $\{\exists x\alpha\} \vdash \alpha$ if x does not occur free in α (i.e. if x is bound in α)

We get many of the same results as before:

Proposition 2.9.2: Properties Of Deduction

1. Let $\varphi_1, \varphi_2, ..., \varphi_n$ be a deduction of $\mathcal{L}$, then $\varphi_1, \varphi_2, ..., \varphi_l$ be a deduction of $\mathcal{L}$ for any l such that $1 \leq l \leq n$.
2. If $\Gamma \vdash \delta$ and $\Gamma \vdash \delta \to \beta$ then $\Gamma \vdash \beta$
3. if $\Gamma \subseteq \Delta$ and $\Gamma \vdash \alpha$ then $\Delta \vdash \alpha$
4. if $\Gamma \vdash \Delta$ and $\Delta \vdash \sigma$ then $\Gamma \vdash \sigma$

Proof **:**
exercise

Theorem 2.9.1: Deduction Theorem

Let Σ be any set of formulas and α, β any formulas. Then $\Sigma \vdash \alpha \to \beta$ if and only if $\Sigma \cup \{\alpha\} \vdash \beta$

Proof **:**
see theorem 2.8.2

We know get a result is a new result due to our axiomatic approach

Theorem 2.9.2: Generalization Theorem

Suppose x is a variable, Γ is a set of formulas in which x does not occur free (i.e. x is either bound or does not exist), and φ is a formula such that $\Gamma \vdash \varphi$. Then $\Gamma \vdash \forall x \varphi$

Proof **:**
If $\varphi \in \Gamma$, then by **A6** we have $\varphi \to \forall x \varphi$, giving us $\forall x \varphi$. If $\varphi \notin \Gamma$, then using induction on the length of the deduction we have a shortest length deduction where the last formula is φ and second last is necessarily $\theta \to \varphi$. Since $\Gamma \vdash \theta \to \varphi$ and we have length less than that of $\Gamma \vdash \theta \to \varphi$, by the induction hypothesis we have $\Gamma \vdash \forall x(\theta \to \varphi)$. With this, we deduce:

(1) $\theta \to \varphi$	
(2) $\forall x(\theta \to \varphi)$	**IH**
(3) $\forall x(\theta \to \varphi) \to (\forall x\theta \to \forall x\varphi)$	**A5**
(4) $(\forall x\theta \to \forall x\varphi)$	2,3 **MP**
(5) $\forall x\theta$	**IH** on $\Gamma \vdash \theta$
(6) $\forall x\varphi$	4,5 **MP**

completing the proof.

Theorem 2.9.3: Generalization On Constants

Suppose that c is a constant symbol, Γ is a set of formulas in which c does not occur, and φ is a formula such that $\Gamma \vdash \varphi$. Then there is a variable x which does not occur in φ such that $\Gamma \vdash \forall x \varphi_x^c$. Moreover, there is a deduction of $\forall x \varphi_x^c$ from Γ in which c does not occur

What this theorem is saying is that if we prove a property of a constant, say x_0 is a constant and $\vdash P(x_0)$, then we in fact have $\vdash \forall x P(x)$

Proof **:**
Similar to the one above

Example 2.9: More Deductions

1. We'll show $\vdash \varphi \to \psi$ then $\vdash \forall x\varphi \to \forall x\psi$. Since x does not occur free in any formula $\emptyset$, it follows from $\vdash \varphi \to \psi$ by the generalization theorem that $\vdash \forall(\varphi \to \psi)$. But then

$$\begin{array}{ll} \forall x(\varphi \to \psi) & \\ \forall x(\varphi \to \psi) \to (\forall x\varphi \to \forall x\psi) & \textbf{A5} \\ \forall x\varphi \to \forall x\psi & \textbf{MP} \end{array}$$

2. show $\vdash \forall x\forall y\forall z(x = y \to (y = z \to x = z))$ (Hint: apply Deduction theorem twice, and show $x = y \vdash y = z$. Now think of the derivation, and finish with Generalization theorem)
3. $\vdash \forall x\alpha \to \exists x\alpha$
4. $\vdash \exists x\gamma \to \forall x\gamma$ if x does not occur free in γ
5. (Using generalization on constants) Let $\mathcal{L}$ be the language of number theory and take $\forall x\exists y\ x < y$. Then we can say given an n, we have $n < n + 1$, so $\exists y\ n < y$, and so we conclude $\forall x\exists y\ x < y$

2.9.1 Soundness and Completeness

Let $\mathcal{L}$ be a fixed first-order language. All formulas will be considered as part of $\mathcal{L}$. First, a set Γ of sentences is inconsistent if $\Gamma \vdash \neg(\psi \to \psi)$ for some formula ψ. A set Σ is maximally consistent of $\Sigma \cup \{\tau\}$ is inconsistent whenever τ is a sentence such that $\tau \notin \Sigma$. Now some review:

Proposition 2.9.3: Results From Earlier

1. (Soundness Theorem) If α is a sentence and Δ is a set of sentences such that $\Delta \vdash \alpha$, then $\Delta \models \alpha$
2. If a set of sentences Γ is satisfiable, then it is consistent
3. Suppose Δ Is an inconsistent set of sentences. Then $\Delta \vdash \psi$ for any formula ψ
4. Supposer Σ is an inconsistent set of sentences. then there is a finite subset Δ of Σ such that Δ is inconsistent
5. A set of sentences Γ Is consistent if and only if every finite subset of Γ is consistent

***Proof* :**
see section 2.8, the proof requires barely any modification.

In propositional logic, we found maximally consistent sets given consistent sets. We may do the same thing in propositional logic, but we have one more tool at our disposal to find maximally consistent sets

Proposition 2.9.4: Relating Structures And Theorems

Let $\mathfrak{M}$ be a structure. Then $Th(\mathfrak{M})$ is a maximally consistent set of sentences

This is the mirror of $\Sigma = \{\varphi \mid v(\varphi) = T\}$

Proof **:**
Define $Th(\mathfrak{M}) = \{\tau \mid \tau \text{ is a sentence and } \mathfrak{M} \models \tau\}$. Let's say $Th(\mathfrak{M})$ is not maximally consistent so that $Th(\mathfrak{M}) \cup \{\tau\}$ is consistent. Necessarily $\mathfrak{M} \not\models \tau$ or else $\tau \in Th(\mathfrak{M})$. But then $\mathfrak{M} \models \neg\tau$ (recall that τ is a sentence, so if $\mathfrak{M} \not\models \tau$ for some assignment, it doesn't do so for any assignment), and so $\neg\tau$ and τ are in $Th(\mathfrak{M}) \cup \{\tau\}$, and the rest of the proof plays out identically.

Example 2.10: Theory Of Structure
Let $\mathfrak{M} = (\{5\})$ be a structure for $\mathcal{L} = \{\}$. Then $Th(\mathfrak{M})$ is a maximally consistent set of sentences. It turns out that

$$Th(\mathfrak{M}) = Th(\{\forall x \forall y\ x = y\})$$

So we also get a set of sentences, $\Sigma = \{\forall x \forall y\ x = y\}$ such that $Th(\Sigma)$ is maximally consistent.

As before, we may abuse maximality to get the same results as in first-order logic, mainly due to the fact we are working with sentences

Proposition 2.9.5: Operational Facts About Maximally Consistent Sets

1. If Σ is a maximally consistent set of sentences, τ is a sentence, and $\Sigma \vdash \tau$, then $\tau \in \Sigma$
2. Suppose Σ is maximally consistent set of sentences and τ is a sentence. Then $\neg\tau \in \Sigma$ if and only if $\tau \in \Sigma$
3. Suppose Σ is a maximally consistent set of sentences and φ and ψ are any sentence. Then $\varphi \in \psi \in \Sigma$ if and only if $\varphi \notin \Sigma$ or $\psi \in \Sigma$
4. Suppose Γ is a consistent set of sentences. Then there is a maximally consistent set of sentences Σ with $\Gamma \subseteq \Sigma$

Proof **:**
these are all repeats of the previous such results. Note that for (4), it is important that our language is countable since we enumerate all sentences in the proof.

Hence, we have the same relation between maximally consistent sets and "propositionally satisfiability". When we did this in propositional logic, we defined a suitable truth assignment function from a maximally consistent set of formulas. Now we need to construct a suitable structure from a maximally consistent set of sentences. The following definition will be key in our construction. The motivation behind this definition is as follows: suppose we have $\Sigma \vdash \exists x\varphi$. Then recall that for $\mathfrak{M}$ to satisfy $\exists x\varphi$, we must have a constant c such that φ_c^x is satisfied by $\mathfrak{M}$. To have $\mathfrak{M} \models \varphi_c^x$, we need $\Sigma \vdash \varphi_c^x$. Thus, if we have $\Sigma \vdash \exists x\varphi$, we in a sense need a constant c such that $\Sigma \vdash \exists x\varphi \to \varphi_c^x$. These constant will be "witnesses" for the existential formulas that our maximally consistent set derives

Definition 2.9.5: Witness

Suppose σ is a set of sentences, and C is a set of (some of the) constants of $\mathcal{L}$. Then C is a *set of witnesses* for Σ in $\mathcal{L}$ if for every formula φ of $\mathcal{L}$ with at most one free variable x, there is a constant symbol $c \in C$ such that $\Sigma \vdash \exists x\varphi \to \varphi^x_c$

Essentially, every element of the universe of formulas for which Σ must exist is "named" or "witnessed" by some constant C, so we take the subset of constant symbols that we use that are associated with symbols $C^{\mathfrak{M}}$ for any structure $\mathfrak{M}$.

Proposition 2.9.6: Extension Of Witnesses

Let Γ and Σ be sets of sentences of $\mathcal{L}$, $\Gamma \subseteq \Sigma$ and C is a set of witnesses for Γ in $\mathcal{L}$. Then C Is a set of witnesses for Σ in $\mathcal{L}$

***Proof* :**
exercise.

Example 2.11: Set Of Witnesses
Let $\mathcal{L}_O$ be the first order language with a signal 2-ary relation symbol $<$ and countably many constant symbols c_q for each $q \in \mathbb{Q}$. Let Σ include all sentences

1. $c_p < c_q$ for every $p, q \in \mathbb{Q}$ such that $p < q$
2. $\forall x(\neg x < x)$
3. $\forall x \forall y(x < y \vee x = y \vee y < x)$
4. $\forall x \forall y \forall z(x < y \to (y < z \to x < z))$
5. $\forall x \forall y(x < y \to \exists z(x < z \wedge z < y))$
6. $\forall x \exists y(x < y)$
7. $\forall x \exists y(y < x)$

Essentially σ asserts that $<$ is a linear order on the universe (points 2 to 4) which is dense (point 5) and has no endpoints (6,7) and has a suborder isomorphic to Q (1). Then $C = \{c_q \mid q \in Q\}$ is a set of Witnesses for Σ in $\mathcal{L}_O$.

With the above example, we can deduce the model for the set of sentences from the set of witnesses by simply letting the universe *be* the set of witnesses, and define the relation intepreting $<$ in a very obvious way from Σ. However, this is due to the example being purposefully well-chosen, in particualr there are not constant symbols which are not witnesses, Σ proves that distinct constant symbols aren't equal to each other, and Σ explicitly has everything you need to know about $<$. In general, we may not know whether Σ has a set of witnesses, for Σ may not even have constants. Furthermore, if Σ has a set of witnesses, say C, we usually can't let the universe just be C (for example, what if $\Sigma \vdash c = d$ with dstinct witnesses c and d). What if have constant symbols whichare

not in C? What if Σ doesnt' prove enough to define relation and functions for the given symbols? Even if Σ does prove enough, how would we be able to tell?

The main way we shall get aroundthis is by workign with maximally consistent sets of sentences in suitable langauges:

Lemma 2.9.2: Generalization on Sentences

Suppose Σ is a set of sentences, φ is any formula and x is any variable. Then $\Sigma \vdash \varphi$ if and only if $\Sigma \vdash \forall x\varphi$

Note that the key part of this lemma is that we are working with sentences.

Proof **:**
By **A6**, we have $\varphi \to \forall x\varphi$, and so we get $\forall x\varphi$.

Theorem 2.9.4: Enough Witnesses

Suppose Γ is a consistent set of sentences of $\mathcal{L}$. Let C be an infinite countable set of constants symbols which are *not* symbols of $\mathcal{L}$ and let $\mathcal{L}' = \mathcal{L} \cup C$ be the language obtained by adding the constant symbols in C to the symbols for $\mathcal{L}$. Then there is a maximally consistent set Σ of sentence $\mathcal{L}'$ such that $\Gamma \subseteq \Sigma$ and C is a set of witnesses for Σ

Proof **:**
here

What this theorem tells us is that we can always brute force a set of witnesses. If there isn't a set of witnesses, we just add one, and if the set doesn't decide enough, we make it decide *everything*.

Theorem 2.9.5: Constructing Structure

Suppose Σ is a maximally consistent set of sentences and C is a set of witnesses for σ. Then there is a structure $\mathfrak{M}$ such that $\mathfrak{M} \models \Sigma$.

Proof **:**
The important thing to do is define $\mathfrak{M}$, the tedious thing to do is show that $\mathfrak{M} \models \Sigma$ but all straight forward.

Using proposition ref:HERE, we get the following corollary

Corollary 2.9.1: Structure And Satisfaction First Order Logic

Suppose Γ is a consistent set of sentences of a first order language $\mathcal{L}$. Then there is a structure $\mathfrak{M}$ for $\mathcal{L}$ satisfying Γ.

Proof **:**
here

with this, we can proceed to prove the soundness theorem as before

Theorem 2.9.6: First Order Consistent Iff Satisfiable

A set of sentences Σ in $\mathcal{L}$ is consistent if and only if it is satisfiable

Proof **:**
same as ref:HERE

Theorem 2.9.7: Completeness Theorem

If α is a sentence and Δ is a set of sentences such that $\Delta \models \alpha$ then $\Delta \vdash \alpha$

Proof **:**
same as ref:HERE

Thus, in first order logic, just like for propositional logic, a sentence is implied by some set of premisses if and only if it has a proof from those premisses

Theorem 2.9.8: Compactness Theorem

A set of sentences Δ is satisfiable if and only if every finite subset of Δ is satisfiable

Proof **:**
same as ref:HERE

2.10 Applications of Compactness

In this chapter, we shall go over some Model theory and some application of the compactness theorem.

Definition 2.10.1: Model Morphism

Let $\mathcal{L}$ be a first-order language and $\mathfrak{M}, \mathfrak{N}$ two models of $\mathcal{L}$. Then:

1. $\mathfrak{M}$ and and $\mathfrak{N}$ are *isomorphic* if there is a funciont $\varphi : |\mathfrak{M}| - |\mathfrak{N}|$ such that
   (a) F is bijective
   (b) $F(c^{\mathfrak{M}}) = c^{\mathfrak{N}}$
   (c) $Ff^{\mathfrak{M}}(a_1, a_2, ..., a_k) = f^{\mathfrak{N}}(F(a_1), ..., F(a_k))$ for every k-placefunction f of $\mathcal{L}$ and $a_1, a_2, ..., a_k \in |\mathfrak{M}$
   (d) $P^{\mathfrak{M}}(a_1, a_2, ..., a_k)$ hodls if and only if $P^{\mathfrak{N}}(F(a_1), ..., F(a_k))$ for every k-place relation symbol of $\mathcal{L}$ and elements $a_1, a_2, ..., a_k \in |\mathfrak{M}$
2. $\mathfrak{M}$ and $\mathfrak{N}$ ar ei elementarily equivalent if $\text{Th}(\mathfrak{M}) = \text{Th}(\mathfrak{N})$, that is if $\mathfrak{M} \models \sigma$ if and only if $\mathfrak{N} \models \sigma$. We writeR $\mathfrak{M} \equiv \mathfrak{N}$

It is easy to see that if $\mathfrak{M} \cong \mathfrak{N}$ then $\mathfrak{M} \equiv \mathfrak{N}$, however the converse need not be true:

Example 2.12: Elementary Equivalent, But Not Isomorphic
Let $\mathfrak{FC} = (\mathbb{N})$ be an infinite model of the basic languaage $\mathcal{L}$. Extend $\mathcal{L}$ to $\mathcal{L}_R$ by adding the constant symbol c_r for every real number r and elt Σ be the set of sentences of $\mathcal{L}$ including

1. every stnence τ of $\text{Th}(\mathfrak{C})$
2. $\neg c_r = c_s$ for eery pair of real numbers r, s such that $r \neq s$

Then every fintie subset of Σ is satisraible. hence by the compatness theorem there is a model $\mathfrak{U}'$ of $\mathcal{L}_R$ satisfying Σ and hence $\text{Th}(\mathfrak{C})$. Let $\mathfrak{U}$ be the model we get by dropping the interpretations of all the constant symbols c_r from $\mathfrak{U}'$ which is then a model of $\mathcal{L}$. Note that $|\mathfrak{U}| = |\mathfrak{U}'|$ since $\mathfrak{U}'$ requires a distinct element of the universe to itnerpret each constant symbol c_r of $\mathcal{L}_R$. Sicne $\text{Th}(\mathfrak{C})$ is maximally consistent set of sentences of $\mathcal{L}$, it follows that $\mathfrak{C} \equiv \mathfrak{U}$. However, $\mathfrak{C}$ cannot be isomorphic to $\mathfrak{U}\mathfrak{K}$ since the cardinalities don't match.

(there are lots of cool examples to add here, especially the non-standard models of the real numbers)

(I would love to add Löwenheim–Skolem theorem and Lefschetz Principle to prove the Ax-Grothendieck Theorem, that is any injective regular function $f : V \to V$ between two varieties over an algebraically closed field (of characteristic zero) is surjective. This utilises those two theorems and compactness)

2.11 Summary

Due to how the course was run for me, I will be summarising this a bit differently for now. I will come back to this another time to add a more comprehensive summary another time

1. **Axiom Schema**: these are the logical axioms:

   A1 $(\alpha \to (\beta \to \alpha))$

A2 $((\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)))$

A3 $(((\neg\beta) \to (\neg\alpha)) \to (((\neg\beta) \to \alpha) \to \beta))$

2. **Modus Ponens**: this is our one rule of inference: $(\{\varphi, (\varphi \to \psi)\}, \psi)$
3. **Deduction** Take a finite sequence of $\mathcal{L}$-formulas $(\varphi_1, \varphi_2, ..., \varphi_n)$. Then this is a *deduction* from Σ if:
   (a) φ_i is an axiom in the axiom schema (i.e. a logical axioms)
   (b) $\varphi_i \in \Sigma$ (i.e. a non-logical axiom)
   (c) φ_k follows from φ_i, φ_j $(i, j < k)$ and modus ponens.
4. **Examples** Look at example 2.5 for a list of important examples, and lemma 2.8.1 for important shortcuts
5. **Contradiction** $\{\delta, \neg\delta\} \vdash \gamma$
6. **Deduction Theorem**
$$\Sigma \vdash \alpha \to \beta \qquad \text{iff} \qquad \sigma \cup \{\alpha\} \vdash \beta$$
7. **More examples** Look at example 2.5., modus Tollens (MT), and corollary 2.8.1 for more important deductions to have at your finger tips.
8. **Soundness Theorem** $\Delta \vdash \alpha$. Then $\Delta \models \alpha$. This one is easy to do with some good inductive reasoning. The rest of the results for propositional deductions builds towards the converse of the Soundness Theorem.
9. **Satisfiability and Consistency**. A set is *inconsistent* if $\Gamma \vdash \neg(\alpha \to \alpha)$ (i.e. it implies a contradiction). If Γ is satisfiable, then Γ is consistent (hence, if *a* truth assignment function works, Γ at least doesn't deduce a contradiction)
10. **Inconsistency**: If inconsistent, then can deduce anything, and only requires finitely many formulas to deduce the inconsistency. The converse of the last statement is that a set is consistent if and only if every finite subset is consistent.
11. **Maximally Consistent**: Σ is maximally consistent if when adding a new formula, Σ becomes inconsistent. Maximally consistent sets really behave like logical objects:
    (a) $\Sigma \vdash \varphi$ then $\varphi \in \Sigma$
    (b) $\neg\varphi \in \Sigma$ if and only if $\varphi \notin \Sigma$
    (c) $\varphi \to \psi \in \Sigma$ if and only if $\varphi \notin \Sigma$ or $\psi \in \Sigma$ (i.e. if $\varphi \notin \Sigma$, it's the vacuous truth case, or if $\varphi \in \Sigma$ then (like in modus ponens) $\psi \in \Sigma$)
12. **Truth Assignment via Maximally Consistent Sets**: Simply do $v(\varphi) = T$ if $\varphi \in \Sigma$ and F otherwise. Then trivially, A maximally consistent set is satisfiable with the above truth assignment function. We next ensure that for any consistent set Γ, we can embed it in a maximally consistent set Σ, and since a set is satisfiable if and only if every subset is satisfiable, we got that *consistent implies satisfiable*. By our earlier result, we have that *consistent if and only if satisfiable*

13. **Completeness Theorem**: $\Sigma \models \varphi$ then $\Sigma \vdash \varphi$.

14. **Compactness Theorem** Γ is satisfiable if and only if every finite subset is satisfiable. This follows form the equivalence with consistency in which this holds.

15. **Generalization**: We know move on to deduction for first-order logic. If φ is any formula and $x_1, x_2, ..., x_n$ any variables, $\forall x_1 ... \forall x_n \varphi$ is the *generalization* of φ. Note that the generalization of a tautology is a tautology.

16. **First order Axiom Schema**:

    (a) **A1** $(\alpha \to (\beta \to \alpha))$

    (b) **A2** $((\alpha \to (\beta \to \gamma)) \to (\alpha \to \beta) \to (\alpha \to \gamma)))$

    (c) **A3** $(((\neg\beta) \to (\neg\alpha)) \to ((\neg\beta) \to \alpha) \to \beta))$

    (d) **A4** $(\forall x\alpha \to \alpha_t^x)$ if t is substitutable for x in α

    (e) **A5** $(\forall x(\alpha \to \beta) \to (\forall x\alpha \to \forall x\beta))$

    (f) **A6** $(\alpha \to \forall x\alpha)$ if x does not occur free in α (i.e. if x is bounded or does not exist)

    (g) **A7** $x = x$

    (h) **A8** $(x = y \to (\alpha \to \beta))$ if α is atomic and β is obtained by replacing some occurrences (possibly all or none) of x in α by y.

17. **Repeat**: deductions and MP are defined the same way. The properties of deductions carry over, and so does the Deduction Theorem

18. **Generalization Theorem**: If x is bound or does not exist in φ and $\Gamma \vdash \varphi$, then $\Gamma \vdash \forall x\varphi$.

19. **Generalization on Constants**: What this theorem is saying is that if we prove a property of a constant, say x_0 is a constant and $\vdash P(x_0)$, then we in fact have $\vdash \forall x P(x)$

20. **More Examples** Looks at 2.9 for more good examples to have at your finger tips.

21. $Th(\mathfrak{M})$ **is maximally consistent**. Define

$$Th(\mathfrak{M}) = \{\tau \mid \tau \text{ is a sentence an } \mathfrak{M} \models \tau\}$$

    Then it is easy to show it is maximally consistent, parallelising our earlier need in propositional deduction for showing that maximally consistent then satisfiable. However, this time we have constants to deal with. Hence

22. **Witness** In first order deductions, we have the existential quantifier, which means that we require a way to "witness" the existence. That's what the set of witnesses does. It is not too hard to show that there is enough witnesses, and that a maximally consistent set Σ along with a set of witnesses C will together be enough to show that there is a $\mathfrak{M}$ such that $\mathfrak{M} \models \Sigma$, that is Σ is satisfiable. The rest follows from the same results almost identical proofs from propositional deduction

3

Computability

In this chapter, we open a new discussion. We are going to characterize a type of function known as a *computable function*. These are functions that, roughly speaking, we can actually "work out" as human being. Since we can work it out, we may try to encode the process into an artificial thinking machine to grind the computation faster (a computer) and get an output for a given input. In other words, computable functions are those for which we can give an input and a set of transformation rules, and in a finite (or countably infinite) amount of time, we can deduce an output. Though it might seem like this is "obvious", mathematics as its now understood does have incomputable functions. For example, $\mathbb{R}$ an $\mathbb{R}^2$ have the same cardinality, which by definition means they are in bijection to each other: $f : \mathbb{R} \to \mathbb{R}^2$. However, it is impossible to define a function where when I give you an input $x \in \mathbb{R}$, and you can tell me $f(x)$ in a finite or even [countably] infinite amount of time. This is because the "construction" of the function used the [uncountable] axiom of choice, which instead of giving a way of choosing each element, it simply asserts that no matter how many sets there are (in our case, $\{x\}$ for each $x \in \mathbb{R}$), we may always choose an element from each set. Due to the existence of uncountable sets (like $\mathbb{R}$), we get some counter-intuitive results which spurred the development of much of logic for the first half of the 20th century.

We are thus going to look at functions in which we can actually find ways of getting values out of them. Our discussion will start with defining what it means to be able to "compute" values. We then use this definition to define computable functions, and then classify all possible computation functions in terms of a small set of initial function that we can use to build all computable functions using composition and two types of recursion. This discussion will lead us to show that in any model of first order logic that tries to capture the rules of arithmetic's, we can always find a contradiction.

3.1 Turing Machines

Fundamentally, a Turing machine tries to capture the idea that we may take an input and manipulate it in some way to get an output using rules of transformation. In essence, a Turing machine will require memory, rules of deduction and a way to take in inputs and give outputs. We start off with the input of a Turing machine. The input will be manipulated by the Turing machine to produce an output.

Definition 3.1.1: Tape

A *tape* is an infinite sequence

$$a = a_0 a_1 a_2 ...$$

such that for each integer i the *cell* $a_i \in \{0, 1\}$. The ith cell is said to be *blank* if a_i is zero and is *marked* if $a_i = 1$. A tape is *blank* if each cell is 0

As a convention, we will assume that all but finitely many cells of any given tape is blank, and that any cell not explicitly described is blank. We will omit as much of the tape as possible, and thus write

$$01011011011$$

to represent the tape 01011011011000..., and we will sometimes put superscript to abbreviate further, for example:

$$01^3 01^{25} 001$$

If σ is any finite sequence, σ^n is $n$copies of the sequence concatenated.

Definition 3.1.2: Tape Position

A *tape position* is a triple $(s, i, \mathbf{a})$ where s and i are natural numbers with $s > 0$, and $\mathbf{a}$ is a tape. Given a tape position $(s, i, \mathbf{a})$, we will refer to cell i as the *scanned cell* and s as the *state*.

We will usually display the tape position by underlining the scanned cell and put the state number to the left. For example, $(2, 4, \mathbf{a})$ can be represented as:

$$2 : 010\underline{1}1001011011$$

that is, it is a tape position at state two with the 4th cell being scanned.

Definition 3.1.3: Turing Machine

A *Turing machine* is a function M such that for some natural number n such that the domain is a subset of

$$\{1, ..., n\} \times \{0, 1\} = \{(s, b) \mid 1 \leq s \leq n,\ b \in \{0, 1\}\}$$

and the range is a subset of

$$\{0, 1\} \times \{-1, 1\} \times \{1, ..., n\}$$

or $\{(c, d, t) \mid c \in \{0, 1\},\ d \in \{-1, 1\},\ 1 \leq t \leq n\}$. We would write the domain and range as Dom and Range so $M : \text{Dom}(M) \to \text{Range}(M)$.

A possible source of confusion is the difference between the input of a Turing machine, the output of a Turing machine, and the tape position. The following summarizes these to diminish confusion:

1. (s, v) is the *state* and the *value* at the tape, so $M(s, v) = (o, d, s')$ maps to ($o$utput, $d$irection, $s$tate) with output values in $\{0, 1\}$, direction values $\{-1, 1\}$ (i.e. left or right) and state value $\{1, 2, ..., n\}$.
2. $(s, c, \mathbf{a})$ is the *state*, the *cell number*, and the *input tape* with $s \in \{1, 2, ..., n\}$ and $c \in \mathbb{N}$.

Often, we shall refer to a Turing machine simply as machine or TM. If $n \geq 1$ is least such that M satisfies the above, we call M an n-state Turing machine and $\{1, ..., n\}$ is the *states* of M. Here is how to interpret M. Let's say $M(s, c) = (b, d, t)$. What M will do is scan at the current cell, overwrite the symbol, move left or right, and execute the next instruction. In particular, if M is in state s (executing instruction number s) and the scanner is presently reading c in cell i, then the machine M should:

1. set $a_i = b$ (replace c with b in the scanned cell)
2. move the scanner to a_{i+d} (move it left if $d = -1$ and right if $d = 1$)
3. enter state t (go to instruction t)

If M is not equipped to handle input c at instruction s (so $M(s, c)$ is undefined), then the computation in progress will stop, or *halt*.

Example 3.1: Turing Machine
We usually visualize Turing machines using table. Take the following

M	0	1
1	$1R2$	$0R1$
2	$0L2$	

where L represents -1 and R represents 1. Then

$$M(1,0) = (1,1,2)$$
$$M(1,1) = (0,1,1)$$
$$M(2,0) = (0,-1,2)$$

and $M(2,1)$ is undefined. Thus we have

$$M\{(1,0),(1,1),(2,0)\} \to \{(1,1,2),(0,1,1),(0,-1,2)\}$$

Let's say the tap on which M was acting on was in position:

$$1 : 010\underline{0}1111o$$

Then, since it was in state 1 while scanning a cell containing 0, it would:

1. write 1 in the scanned cell
2. more the scanner one cell to the right
3. go to state 2

Since M doesn't know what to do with input 1 in state 2, it would halt, ending the computation

This leads us to ask questions of *computability*. Informally, a computation is a sequence of actions of a machine M on a tape according to the given rules described above, starting with instruction 1 and the scanner at cell a_1 (or a_0 if you're a computer scientist) on the given tape. A computation ends (*halts*) when and if the machine encounters a tape position which it doesn't know what to do in. If it never halts, and doesn't *crash* by running the scanner off the left and of the tape[1], the computation never ends. We make this notion rigorous

Definition 3.1.4: Computation

Let M be a Turing machine. Then

1. if $p = (s,i,\mathbf{a})$ is a tape position and $M(s,a_i) = (b,d,t)$ is defined, then $\mathbf{M}(p) = (t, i+d, \mathbf{a}')$ is the *successor tape position* where $a'_i = b$ and $a'_j = a_j$ whenever $j \neq i$
2. A *partial computation* with respect to M is a sequence $p_1p_2...$ of tape positions such that $p_{l+1} = \mathbf{M}(p_l)$ for each $l < k$.
3. A partial computation $p_1, p_2, ..., p_k$ with respect to M is a *computation* (with respect to M) with *input tape* $\mathbf{a}$ if $p_1 = (1,0,\mathbf{a})$ and $\mathbf{M}(p_k)$ is undefined (and *not* because the scanner run off the end of the tape). The *output tape* of the computation is the tape of the final tape position p_k.

As we've defined things, a partial computation is a computation only if the Truing machine halts but doesn't crash in the final tape position. The requirement that it halts means that any computation can only have finitely many steps. Unless stated otherwise, all partial computations on a given input

[1] some author's treat this as another way of haling

begins in state 1.

Example 3.2: Computations

1. Let M be the same Turing machine as in example 3.1. Let $\mathbf{a} = 1100$. Then the initial starting point must be
$$1 : \underline{1}100$$
Then going trough the computations, we get:
$$\begin{aligned} 1 &: 0\underline{1}00 \\ 1 &: 00\underline{0}0 \\ 1 &: 001\underline{0} \\ 1 &: 0\underline{1}0 \end{aligned}$$
Since $M(2,1)$ is undefined, the process terminates, hence we have given a computation.
2. (look at homework and quiz for further examples)

It will be useful for future purposes to have a library of Turing machines that manipulate blocks of 1s in various ways and very useful to be able to combine machines performing simpler tasks to perform more complex ones. (give such Turing machines here)

As we've defined Turing machines, we have set ourselves many (potential) limitations. For example, why only input 0 and 1? Why have only one tape? Here are a list of possible additions and expansions we may wish to add to our definition:

1. The machines can move the scanner many steps at a time
2. any finite alphabet of at least two symbols can e use
3. Separate the read and write heads
4. have multiple heads
5. have the tape be two-way infinite
6. have multiple tapes
7. have two or higher dimensional tapes

We shall see that anything that is computable on any of the possible augmentations of Turing machines given from any combinations of the above features is in fact computable on a Turing machine, and so in terms of computability these augmentations are "redundant" (they don't add any extra computations).

(I skipped this cause its routine checks. I looked over it, and if I need to in the future I'll write about it)

From now on, we shall assume we may have a Turing machine augmented with any of the above additions, since we have shown they are all simulated by Turing Machines.

3.2 Computable Functions and Relations

We shall discuss what it means to have a computable function and computable relations. From here on out, functions may be *partial functions*, in which case we shall explicitly specify what is the domain and codomain of f on which f becomes a function, or write $f : \text{Dom}(f) \to Y$. Note that any k-place function is a particular $k+1$-place relation:

$$R_f(n_1, n_2, ..., n_k, m) \quad \Leftrightarrow \quad f(n_1, n_2, ..., n_k) = m$$

and if R is a k-place relation, there is an induced function known as the *characteristic function* of R defined by:

$$\chi_R(n_1, n_2, ..., n_k) = \begin{cases} 1 & \text{if } R(n_1, n_2, ..., n_k) \text{ is true} \\ 0 & \text{if } R(n_1, n_2, ..., n_k) \text{ is false} \end{cases}$$

Natural numbers on tapes will be represented in unary notation: n will be represented as 1^{n+1} (the extra 1 serves as "memory" for the Turing machine to be aware of some of it's actions). If $(n_1, n_2, ..., n_k) \in \mathbb{N}^k$, then this will be represented as

$$1^{n_1+1}01^{n_2+1}...01^{n_k+1}$$

This way of defining natural numbers on a tape is not terribly space efficient (think of binary representations), but it will be Turing-machine-computation friendly.

Definition 3.2.1: [Turing] Computable Function

Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial function. Then f is said to be *[Turing] computable* (or just *computable*) if for any $(n_1, n_2, ..., n_k) \in \text{Dom}(f)$, the computation of M with input tape

$$\underline{0}1^{n_1+1}01^{n_2+1}...01^{n_k+1}$$

will halt with output tape

$$\underline{0}1^{f(n_1,n_2,...,n_k)+1}$$

A machine M that satisfies the above condition is said to *compute* f.

Example 3.3: Computable Functions

1. Show that $\iota : \mathbb{N} \to \mathbb{N}$ defined by $\iota(n) = n$ is computable with machine $M = \emptyset$.
2. Show that $f(n, m) = n + m$ is computable
3. show that $\pi_1 : \mathbb{N}^2 \to \mathbb{N}$ giving $\pi(n, m) = n$ is computable
4. show that $O : \mathbb{N} \to \mathbb{N}$ defined by $O(n) = 0$ is computable.
5. Show that

$$PRED(n) = \begin{cases} n-1 & n \geq 1 \\ 0 & n = 0 \end{cases}$$

is computable.

6. Show that
$$Diff(n) = \begin{cases} n-m & n \geq m \\ 0 & n < m \end{cases}$$
is computable.

7. show that any projection functions $\pi_i : \mathbb{N}^k \to \mathbb{N}$ is computable.

Note that not all functions $f : \mathbb{N}^k \to \mathbb{N}$ are computable, even when $k = 1$. To see this, note that there are $\mathbb{N}^{\mathbb{N}}$ number of functions from $\mathbb{N}$ to $\mathbb{N}$, i.e. uncountably many. By definition, a Turing machine has a finite domain and codomain. All possible Turing machines over a particular finite domain and codomain is countable, and the union of all such machines is a countable union which (by the countable axiom of choice) is still countable. Hence, there must be non-computable functions. Hence, there are more unique functions of the form $f : \mathbb{N} \to \mathbb{N}$ than there are unique Turing machines to show that each function is computable.

For a concrete example (which will be given, but not proven), consider the following:

Definition 3.2.2: Busy Beaver Competition

A machine M is an *n-state entry in the busy beaver competition* if:

1. M has a two-way infinite tape and alphabet $\{1\}$[a]
2. M has $n+1$ states, but state $n+1$ is used only for halting ($M(n+1,0)$ and $M(n+1,1)$ is undefined)
3. M eventually halts when given a blank input tape

The *score* of M in the competition is the number of 1's on the output tape of its computation from a blank input tape. The maximum of all possible scores for n-state entry machine is denoted $\Sigma(n)$

[a]Remember that such a machine can be converted to a Turing Machine

The function $\Sigma(n)$ is well-defined as there are only finitely many Turing $n+1$ state machines (with alphabet $\{1\}$) and since we are taking the subset which must halt, a maximum must be achieved.

Example 3.4: Busy Beaver Competition

1. The only 0-state entry is $M = \emptyset$, and so trivially $\Sigma(0) = 0$
2. Define P as

P	0	1
1	1R2	1L2
2	1L1	1L3

Then P is a 2-tate entry in the busy beaver competition with a score of 4, so $\Sigma(2) \geq 4$.

3. Show that $\Sigma(1) = 1$
4. Show that $\Sigma(3) \geq 6$
5. Show that $\Sigma(n) < \Sigma(n+1)$

The function Σ grows very quickly. We know that:

$$\Sigma(0) = 0 \quad \Sigma(1) = 1 \quad \Sigma(2) = 4 \quad \Sigma(3) = 6 \quad \Sigma(4) = 13$$

however, $\Sigma(5)$ is unknown (we know it's at least 4098, which takes around 40 million steps).

Proposition 3.2.1: Non-Computable Function

The function Σ is *not* computable by any Turing machine

***Proof* :**
see T. Rado, On non-computable functions, Bell System Tech. J. 41 (1962), 877– 884.

Coming back to computational functions, we have that they are very flexible:

Proposition 3.2.2: Composition Is Turing Computable

Let f and g be computable functions such that $g \circ f$ is well-defined. Then $g \circ f$ is computable

***Proof* :**
For simplicity, let's say $f : \mathbb{N}^1 \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$. Consider $(g \circ f)(x) = g(f(x))$. By definition of f, there exists a Turing machine, M_f such that when translating x into unary representation, M_f halts on

$$01^{f(x)+1}$$

Since $g \circ f$ is well-defined, $f(x)$ is a valid input for g. Then by the computability of g, there exists a Turing machine M_g such that when translating $f(x)$ into unary representation, M_g halts on

$$01^{g(f(x))+1} \tag{3.1}$$

Thus, we see that we can modify M_f so that instead of halting on the last input, it instead goes to the initial state of M_g. We may combine these two Turing machines and label it $M_{g\circ f}$. Thus, we have a Turing machine such that on input x in unary representation, we get that $M_{g\circ f}$ halts[a] on the tape at the position in equation (3.1). Since this is true for arbitrary input, we see that $g \circ f$ must be computable, completing the proof.

[a] in finite amount of time. For us, halting has to happen in a finite amount of time

With computable functions being closed under function composition, we may imagine there is a "basis" of computational functions:

Definition 3.2.3: Initial Functions

The following are called the *initial functions*

1. O, the 1-place function such that $O(n) = 0$ for all $\mathbb{N}$
2. S, the 1-place function such that $S(n) = n + 1$
3. $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$ that projects onto the ith component (note that π_1^1 is the identity)

Note that we will be allowed to combine a finite number of inputs and still consider it a composition:

$$(O, S) : \mathbb{N}^2 \to \mathbb{N}^2 \qquad (a, b) \mapsto (0, b + 1)$$

Though composing these functions gives us a good family of functions, we are nevertheless limited in the number of computable functions we have by this method as we can see by this next proposition:

Proposition 3.2.3: Composition Of Initial Functions Is Bounded

Let f be a 1-place function obtained from finitely many compositions of initial functions. Then there exists a constant $c \in \mathbb{N}$ such that $f(n) \leq n + c$

Proof **:**
easy to do inductively

This shows us that there are in fact relatively few computable functions developed through composition. We thus will define 2 more methods by which we may construct computable functions which allows for a much greater class of them. We'll show these 3 methods will allow us to construct all computable function on $\mathbb{N}$ from the initial functions.

3.2.1 Primitive Recursion

Primitive recursion is what we would usually call recursion in computer science or induction in mathematics. We call it primitive recursion to distinguish it with another recursive method to define functions which we will define in the next section (unbounded minimalization). Primitive recursion will be sufficient to build up essentially all familiar arithmetic functions from initial functions

Definition 3.2.4: Primitive Recursion

Let g be a k-place function ($k \geq 1$), and h a $(k + 2)$-place function. Let f be a $(k + 1)$-place function such that

1. $f(n_1, n_2, ..., n_k, 0) = g(n_1, n_2, ..., n_k)$
2. $f(n_1, n_2, ..., n_k, m + 1) = h(n_1, n_2, ..., n_k, m, f(n_1, n_2, ..., n_k, m))$

for every $(n_1, n_2, ..., n_k) \in \mathbb{N}^k$ and $m \in \mathbb{N}$. Then f is said to be obtained from g and h by *primitive recursion.*

We see how this defines f inductively: it's initial value is given by g, and the rest is given by h with the same inputs as well as the preceding value of f. Note that if f is a 1-place function, then we define it by doing:

1. $f(0) = c$ for some constant c
2. $f(n+1) = (n, f(n))$

Example 3.5: Primitive Recursion

1. $SUM(n,m) = n + m$ can be obtained by primitive recursion from the initial function π_1^1 and $S \circ \pi_3^3$ of initial functions as follows:
   (a) $SUM(n,0) = \pi_1^1(n)$
   (b) $SUM(n,m+1) = (S \circ \pi_3^3)(n,m,SUM(n,m))$

   We may prove this with induction, the base case is $SUM(n,0) = \pi_1^1(n) = n = n + 0$. and for $m \geq 0$, assume $SUM(n,m) = n + m$ so that then

$$\begin{aligned} SUM(n,m+1) &= (S \circ \pi_3^3)(n,m,SUM(n,m)) \\ &= S(\pi_3^3(n,m,SUM(n,m))) \\ &= S(SUM(n,m)) \\ &= SUM(n,m) + 1 \\ &= n + m + 1 \end{aligned}$$

2. $h(n_1, n_2, ..., n_k, m) = \sum_{i=0}^{m} g(n_1, n_2, ..., n_k, i)$ for any $k \geq 0$ and each g is $(k+1)$-place primitive recursive
3. $MULT(n,m) = nm$ is obtained via primitive recursion on O and $SUM \circ (\pi_3^3, \pi_1^3)$ by defining
   (a) $MULT(n,0) = O(n)$
   (b) $MULT(n,m+1) = (SUM \circ (\pi_3^3, \pi_1^3))(n,m,MULT(n,m))$

   Then:

$$\begin{aligned} MULT(n,m+1) &= (SUM \circ (\pi_1^3, \pi_3^3))(n,m,MULT(n,m)) \\ &= SUM(n, MULT(n,m)) \\ &= SUM(n,nm) \\ &= n + nm \\ &= (1+m)n \\ &= n(1+m) \end{aligned}$$

   giving us $MULT(n,m+1) = n(m+1)$.
4. $EXP(n,m) = n^m$. Define $EXP(x,0) = 1$ and

$$EXP(x,n+1) = (MULT \circ (\pi_1^3, \pi_3^3))(x,n,EXP(x,n))$$

where $MULT(n,m) = nm$. Given for a moment that $MULT$ is well-defined, we compute:

$$\begin{aligned} EXP(x,n+1) &= MULT((\pi_1^3, \pi_3^3)(x,n,EXP(x,n))) \\ &= MULT(x, EXP(x,n)) \\ &= MULT(x, x^n) \\ &= xx^n \\ &= x^{n+1} \end{aligned}$$

giving us $EXP(x,n+1) = x^{n+1}$, which is our desired definition. If $MULT(n,0) = 0$ and then

5. For $FACT(n) = n!$, define:

$$Fact(n) = \begin{cases} 1 & n = 0 \\ MULT(n, Fact(n-1)) & n > 0 \end{cases}$$

6. The function $PRED(n) = n - 1$ is primitive recursive since

$$PRED(n+1) = \begin{cases} O(n) & n = 0 \\ \pi_1^2(n, PRED(n)) & n > 0 \end{cases}$$

7. Recall

$$DIFF(n,m) = \begin{cases} n - m & n \geq m \\ 0 & \text{otherwise} \end{cases}$$

For a fixed n where $n \geq m$, we may equivalently think of this function as $D_n(m)$. Then:

$$D_n(m) = \begin{cases} n & m = 0 \\ PRED(D_n(PRED(m))) & m > 0 \end{cases}$$

Note that when $n < m$, we get $PRED(D_n(m)) = PRED(0) = 0$. Thus, define

$$DIFF(n,m) = D_n(m)$$

for each $n \in \mathbb{N}$.

8. Define

$$AbsDiff(n,m) = |n - m|$$

We will write $n \dot{-} m$ for $DIFF(n,m)$. First, note that:

$$|n-m| = (n \dot{-} m) + (m \dot{-} n) = SUM((n \dot{-} m), (m \dot{-} n))$$

and that

$$m \dot{-} n = \pi_2^2(n,m) \dot{-} \pi_1^2(n,m)$$

Thus:

$$AbsDiff(n,m) = SUM(n \dot{-} m, \pi_2^2(n,m) \dot{-} \pi_1^2(n,m))$$

Proposition 3.2.4: Computable Functions Closed Under Primitive Recursion

Let g be a computable k-place function ($k \geq 1$), h a computable $(k+2)$-place function. Then if f is obtained from g and h by primitive recursion, then f is also Turing computable

***Proof* :**
The inputs $f(n_1, n_2, ..., n_k, 0)$ are all computable by definition. Each input $f(n_1, n_2, ..., n_k, m)$ is computable by a composition of Turing Machines. The key idea is to have 3 tapes: one for remembering the input $(n_1, n_2, ..., n_k)$, one for the index m, and one for the input $f(n_1, n_2, ..., n_k)$.

Definition 3.2.5: Primitive Recursive Function

Let $f : \mathbb{N}^k \to \mathbb{N}$ be a function. Then f is said to be *primitively recursive* if it can be defined from the initial functions by finitely many compositions and primitive recursion.

Lemma 3.2.1: Important Primitive Recursive Functions

Show that the following functions are primitive recursive

1. For any $k \geq 0$ and primitive recursive $(k+1)$-place function g, the $(k+1)$-place function f given by
$$f(n_1, n_2, ..., n_k, m) = \prod_{i=0}^{m} g(n_1, n_2, ..., n_k, i)$$
is primitive recursive
2. For any constant $a \in \mathbb{N}$,
$$\chi_{\{a\}}(n) = \begin{cases} 0 & n \neq a \\ 1 & n = a \end{cases}$$
is primitive recursive
3. If f is a primitive recursive k-place function, and $a, c_1, c_2, ..., c_k \in \mathbb{N}$, the function
$$h(n_1, n_2, ..., n_k) = \begin{cases} f(n_1, n_2, ..., n_k) & (n_1, n_2, ..., n_k) \neq (c_1, c_2, ..., c_k) \\ a & (n_1, n_2, ..., n_k) = (c_1, c_2, ..., c_k) \end{cases}$$
is primitive recursive

***Proof* :**

1. Define
$$f(n_1, n_2, ..., n_k, 0) = g(n_1, n_2, ..., n_k, 0)$$
Next, we shall overload the notation of π_i^j so that $\pi_1^3, \pi_2^3, \pi_3^3$ will represent projecting $(n_1, n_2, ..., n_k, m, f(n_1, n_2, ..., n_k, m))$ to $(n_1, n_2, ..., n_k), m,$ and $f(n_1, n_2, ..., n_k, m)$ respec-

tively. Then:

$$\begin{aligned} f(n_1, n_2, ..., n_k, m+1) &= (Mult \circ (g \circ (\pi_1^3, S(\pi_2^3)), \pi_3^3))(n_1, ..., n_k, m, f(n_1, ..., n_k, m)) \\ &= Mult \circ \left(g(n_1, ..., n_k, m+1), \prod_i^m g(n_1, ..., n_k, i) \right) \\ &= g(n_1, ..., n_k, m+1) \prod_i^m g(n_1, ..., n_k, i) \\ &= \prod_i^{m+1} g(n_1, n_2, ..., n_k, i) \end{aligned}$$

giving us our function

2. First, define

$$\chi_{\{0\}}(n) = \begin{cases} 1 & n = 0 \\ O(n-1) & n > 0 \end{cases}$$

Then, for $\chi_{\{a\}}$, define

$$\chi_{\{a\}}(n) = \chi_{\{0\}}(|n - a|)$$

see example 3.5(7) for $|n - a| = AbsDiff(n, a)$.

3. We may first modify $\chi_{\{0\}}$ to χ_0^a so that:

$$\chi_0^a = \begin{cases} 1 & n = 0 \\ C_a(n) & n > 0 \end{cases}$$

Then, we augment $\chi_{\{c\}}$ to $\chi_{\{(n_1, n_2, ..., n_k)\}}$ by doing

$$\chi_{\{(n_1, n_2, ..., n_k)\}}(m_1, m_2, ..., m_k) = Mult(Mult(\cdots Mult(\chi_{\{n_1\}}(m_1), \chi_{\{n_2\}}(m_2)), \chi_{\{n_3\}}(m_3)), \cdots)\chi_{\{n_k\}}(m_k))$$

Then:

$$h(n_1, n_2, ..., n_k) = Sum(Mult(\chi_c, a), Mult(Diff(1, \chi_c), f))(n_1, n_2, ..., n_k)$$

Which translates to:

$$h(n_1, n_2, ..., n_k) = \begin{cases} f(n_1, n_2, ..., n_k) & (n_1, n_2, ..., n_k) \neq c \\ a & (n_1, n_2, ..., n_k) = c \end{cases}$$

Theorem 3.2.1: Primitive Recursive, Then Computable

The set of all primitive recursive functions is computable.

***Proof* :**
exercise

We may extend the notion of primitive recursive to relations using our earlier observation of how all relations can be represented through characteristic functions:

Definition 3.2.6: Primitive Recursive Relation

Let $k \geq 1$. Then a k-place relation $P \subseteq \mathbb{N}^k$ is *primitive recursive* if the characteristic function

$$\chi_P(n_1, n_2, ..., n_k) = \begin{cases} 1 & (n_1, n_2, ..., n_k) \in P \\ 0 & (n_1, n_2, ..., n_k) \notin P \end{cases}$$

is primitive recursive

Almost all relations we've seen are primitive recursive:

Example 3.6: Primitive Recursive Relations and further Functions

1. $P = \{2\} \subseteq \mathbb{N}$ is a primitive recursive relation since $\chi_{\{2\}}$ is recursive.
2. If P is primitive recursive, $\neg P$ (i.e. $\mathbb{N}^k \setminus P$) is primitive recursive. recursive. Notice we can take the compliment of the characteristic to define $\chi_{\mathbb{N}^*}(n)$ where $\mathbb{N} = \mathbb{N} \setminus \{0\}$ by

   $$\chi_{\mathbb{N}^*}(n) := \chi_{\{0\}}(\chi_{\{0\}}(n))$$

   Hence, if P is primitive recursive:

   $$\chi_{\neg P}(n) = \chi_0(\chi_P(n))$$

3. $P \vee Q$ (i.e. $P \cup Q$) is primitive recursive if P, Q are primitive recursive
4. $P \wedge Q$ (i.e. $P \cap Q$) if P, are primitive recursive
5. $EQUAL$ where $EQUAL(n, m) \Leftrightarrow n = m$
6. DIV, where $DIV(n, m) \Leftrightarrow n \big| m$. First, define $mod(0, m) = 0$ and

   $$mod(n+1, m) = \begin{cases} 0 & mod(n, m) + 1 = m \\ mod(n, m) + 1 & mod(n, m) + 1 \neq m \end{cases}$$

   This defines the mod function. Hence

   $$DIV(n, m) = \begin{cases} 1 & mod(n, m) = 0 \\ 0 & mod(n, m) \neq 0 \end{cases}$$

7. $ISPRIME$, where $ISPRIME(n) \Leftrightarrow n$ is prime. The key is to count the number of divisor. If c is a count of divisors, then $c(p) = 2$ if p is prime. Define

   $$f(n, m) = \begin{cases} 0 & m = 0 \\ \sum_{k=1}^{m} div(n, k) & m > 0 \end{cases}$$

which is primitive recursive since summation is primitive recursive and *div* is primitive recursive. Then

$$c(n) = f(n, n)$$

Then

$$\chi_{\text{Prime}}(n) := (c(n) = 2) := \chi_{=}(c(n), 2)$$

8. Let $p(n) = n$th prime. The base case of $n = 2$ is easy. For the induction, we will use a proposition we have yet to prove known as *bounded mimialization*, see proposition 3.2.6. Let $p(n)$ be the n-prime number, and we want to show that $p(n+1)$ returns the next smallest prime number. Take the interval $\{y \in \mathbb{N} \mid f(n) \leq f(n)! + 1\}$. This is a bounded set which we can see certainly has the next smallest prime number[a], hence we may take a function f on this interval and ask it to return to us the *least* prime number (by the above, checking if a number is prime is primitive recursive). By proposition 3.2.6, this is a primitive recursive relation, and hence $p(n+1) = \mu(f(n))$ where μ represents the "minimizing" we've just done.
9. $POWER(n, m) = k$ where $k \geq 0$ is maximal such that $n^k \big| m$.
10. $LENGTH(n) = \ell$ where ℓ is maximal such that $p_\ell \big| n$
11. $ELEMENT(n, i) = n_i$ if $n = p_1^{n_1} \cdots p_k^{n_k}$ (and $n_i = 0$ if $i > k$)
12. Take
$$SUBSEQ(n, i, j) = \begin{cases} p_i^{n_i} p_{i+1}^{n_{i+1}} \cdots p_j^{n_j} & 1 \leq i \leq j \leq k \\ 0 & \text{otherwise} \end{cases}$$
whenever $n = p_1^{n_1} \cdots p_k^{n_k}$
13. $CONCAT(n, m) = p_1^{n_1} \cdots p_k^{n_k} p_{k+1}^{m_1} \cdots p_{k+\ell}^{m_\ell}$ if $n = p_1^{n_1} \cdots p_k^{n_k}$ and $m = p_1^{m_1} \cdots p_\ell^{m_\ell}$

[a] exercise if you don't see it

The above collection of primitive recursive functions and relation give us the tools for representing finite sequences of integers by a single integer, as well as tools for manipulating these representations. Thus, at least in principle, we may reduce all problems with primitive recursive functions and relations to problem involving only 1-place primitive recursive functions and relations

Theorem 3.2.2: K-Place To 1-Place Primitive Recursive Function

A k-place function g is primitive if and only if the 1-place function h given by $h(n) = g(n_1, n_2, ..., n_k)$ if $n = p_1^{n_1} \cdots p_k^{n_k}$ is primitive recursive.

***Proof* :**
Let G be the k-place function and g be the 1-place function. Let's say G is primitive recursive, so that we want to show that g is primitive recursive. Since $g(p_1^{n_1} \cdots p_k^{n_k}) = G(n_1, n_2, ..., n_k)$, Define

$$g(p_1^{n_1} p_2^{n_2} \cdots n_k^{n_k}) = G(pow(p_1, n), ..., pow(p_k, n))$$

Since *pow* and G is primitive recursive, so is their composition, so g is primitive recursive.

Conversely, let's say g is primitive recursive so that we want to show that G is primitive recursive.

Then:

$$G(n_1, n_2, ..., n_k) = g\left(\prod_i^k exp(p(i), n_i)\right)$$

since $\prod$, exp, p, and g are primitive recursive, their composition is primitive recursive, so G is primitive recursive, completing the proof.

Corollary 3.2.1: K-Place To 1-Place Primitive Recursive Relation

A k-place relation g is primitive if and only if the 1-place relation P' is primitive recursive, where

$$(n_1, n_2, ..., n_k) \in P \Leftrightarrow p_1^{n_1} \cdots p_k^{n_k} \in P'$$

***Proof* :**
This comes down to translating relations to primitive recursive characters, where then we can apply the above theorem.

Note that not all computable functions are primitive recursive function:

Definition 3.2.7: Ackerman's Function

Define the 2-place function A as follows:

1. $(0, \ell) = S(\ell)$
2. $A(S(k), 0) = A(k, 1)$
3. $A(S(k), S(\ell)) = A(k, A(S(k), \ell))$

Given A, define the 1-place function α by $\alpha(n) = A(n, n)$. It is not too hard to see that A, and hence α are Turing computable.

Ackermann function essentially stack power towers. Consider for simplicity, consider $\alpha_4(n) = A(4, n)$. Then:

$$\begin{aligned}
\alpha_4(0) &= 2^0 = 1 \\
\alpha_4(1) &= 2^1 = 2 \\
\alpha_4(2) &= 2^2 = 4 \\
\alpha_4(3) &= 2^{2^2} = 16 \\
\alpha_4(4) &= 2^{2^{2^2}} = 65536 \\
\alpha_4(5) &= 2^{2^{2^{2^2}}} = 2^{65536} \\
&\vdots
\end{aligned}$$

Proposition 3.2.5: Ackerman's Functions Growth

Let α be Ackerman's function and f be any primitive recursive function. Then there is an $n \in \mathbb{N}$ such that for all $k > n$, $\alpha(k) > f(k)$. As a consequence, α is not primitive recursive.

Proof **:**
exercise.

3.2.2 Unbounded Minimalization

Our final method will round off our tool box of possible ways of constructing computable functions. We shall show that using composition, primitive composition, and Unbounded Minimalization, we will have all Computable functions.

Definition 3.2.8: Unbounded Minimalization

Let g be a $(k+1)$-place function $(k \geq 1)$. Then the *unbounded minimalization* of g is the k-place function f defined by

$$f(n_1, n_2, ..., n_k) = m \text{ where } m \text{ is least so that } g(n_1, n_2, ..., n_k, m) = 0$$

We often denote this as

$$f(n_1, n_2, ..., n_k) = \mu m[g(n_1, n_2, ..., n_k, m) = 0]$$

If no such m exists, then we say the unbounded minimalization of g is not defined on $(n_1, n_2, ..., n_k)$.

Note that it is not unbounded minimalization was not defined algorithmically, but logically. In other words, it is not obvious if unbounded minimalization is computable. If a give you a function g, can you determine in a finite amount if the unbounded minimalization of g is well-defined? We thus need to guarantee our function always can terminate

Definition 3.2.9: Regular Function

A $(k+1)$-place function g is said to be *regular* if for every $(n_1, n_2, ..., n_k) \in \mathbb{N}^k$, there is at least one $m \in \mathbb{N}$ such that $g(n_1, n_2, ..., n_k, m) = 0$.

Hence we may do the obvious algorithm of computing $g(n_1, n_2, ..., n_k, m)$ for $m = 0, 1...$, until we find an m so that $g(n_1, n_2, ..., n_k, m) = 0$.

Example 3.7: Unbounded Minimalization

1. $f(n)$ is the function that returns the smallest prime factor of n
2. If $p|n$, then $g(n) = k$ where $n = pk$.
3. $p(n)$ returns the nth prime number.

The reason why this is called unbounded minimization is due to the following

Proposition 3.2.6: Bounded Minimalization

Let g be a $(k+1)$-place primitive recursive regular function such that for some primitive recursive k-place function h,

$$\mu m[g(n_1, n_2, ..., n_k, m) = 0] \leq h(n_1, n_2, ..., n_k)$$

for all $(n_1, n_2, ..., n_k) \in \mathbb{N}$. Then $\mu m[g(n_1, n_2, ..., n_k, m) = 0]$ is primitive recursive

***Proof* :**
exercise

Proposition 3.2.7: Unbounded Minimalization Is Computable

Let g be a Turing computable regular $(k+1)$-place function. Then the unbounded minimalization of g is Turing computable.

***Proof* :**
Let $\mu_m^g(n_1, n_2, ..., n_k, m) = \mu m[g(n_1, n_2, ..., n_k, m) = 0]$. We want to show μ_m^g halts on all inputs. Since g is computable, for any input it will halt. Fixing a particular $(n_1, n_2, ..., n_k)$ and considering M_l for $m = l$, since g is regular, We know M_l halts. We may then concatenate it with a machine that checks if the output is zero.

finish here

We now combine all our types of functions

Definition 3.2.10: Recursive Function

Let f be a k-place function. Then f is *recursive* if it can be defined from the initial functions by finitely many application of composition, primitive recursion, and unbounded minimalization of regular functions.

Similarly, a k-place partial function is *recursive* if it can be defined from initial functions by finitely many application of composition, primitive recursion, and unbounded minimalization of (possibly non-regular) functions.

Definition 3.2.11: Recursive Relation

A k-place relation P is said to be *recursive* (or Turing computable) if its characteristic function χ_P is recursive

Theorem 3.2.3: Recursive Function, Then Computable

Let f be a recursive function. Then f is computable

Proof **:**
This is combining all our previous results.

In the next section, we shall show the following very important result

Theorem 3.2.4: Computable, Then Recursive

Let f be a computable function. Then f is recursive

Hence, recursive and Turing computable is synonymous. We end this section with some important technical results

Theorem 3.2.5: K-Place To 1-Place Recursive Function

A k-place function g is recursive if and only if the 1-place function h given by $h(n) = g(n_1, n_2, ..., n_k)$ if $n = p_1^{n_1} \cdots p_k^{n_k}$ is recursive.

Proof **:**
This proof is very similar to theorem 3.2.2.

Corollary 3.2.2: K-Place To 1-Place Recursive Relation

A k-place relation P is recursive if and only if the 1-place relation P' is recursive where

$$(n_1, ..., n_k) \in P \qquad \Leftrightarrow \qquad p_1^{n_1} \cdots p_k^{n_k} \in P'$$

Proof **:**
immediate consequence of above theorem using characteristic functions.

3.3 Characterizing Computability

In this section, we work towards showing that all computable functions are recursive. We will show that any Turing machine can be simulated by some recursive function. Since the functions we worked with input natural numbers, we will have to codify the tape and the Turing machine as natural numbers.

Definition 3.3.1: Code of Position

Let $(s, i, \mathbf{a})$ be a tape position such that all but finitely many cells of $\mathbf{a}$ are blank. Let $n \in \mathbb{N}$ such that $a_k = 0$ for all $k > n$. Then the *code* of $(s, i, \mathbf{a})$ is

$$[(s, i, \mathbf{a})] = 2^2 3^i 5^{a_0} 7^{a_1} 11^{a_2} \cdots p_{n+3}^{a_n}$$

For example,

Example 3.8: Code of Position

1. the code of $(2, 1, 1001)$ is:

$$[(2, 1, 1001)] = 2^2 3^1 5^1 7^0 11^0 13^1 = 780$$

2. Let $(1, 0, \mathbf{a})$ where $\mathbf{a}$ is blank. Then it's code is 2
3. Let $(4, 3, \mathbf{a})$ where $\mathbf{a}$ is 1011100101. What is its code?
4. What tape position do we get from 10314720. What is the tape position?

Definition 3.3.2: Code Of Sequence Of Tape Positions

Let $t_1, t_2, ..., t_n$ be a sequence of tape positions. Then the code of the sequence is

$$[t_1, t_2, ..., t_n] = 2^{[t_1]} 3^{[t_2]}$$

Note that both the tape positions and the sequence of tape positions have unique codes, and hence they are injective. However, they are *not* surjective, for example there is no code $2^{2^1 3^1 5^2}$ since there cannot be 2 "1"'s in the first cell. Hence, detecting when a number is a tape position or a sequence of tape positions is important, and for our purposes it has to be done in a recursive way:

Proposition 3.3.1: Tapepos And Tapeposseq

Define the following two relations:

1. **TapePos**, where **TapePaos**$(n) \Leftrightarrow n$ is the code of a tape position
2. **TapePosSeq**, where **TapePoSeq**$(n) \Leftrightarrow n$ is the code of a sequence of the tape position

show that these two relations are primitive recursive.

***Proof* :**

1. A tape position is a triple $(s, i, \mathbf{a})$ where $i \in \mathbb{N}$ and $s > 0$. Hence, define:

$$TapePos(n) = div(2, n) \cdot \left(\prod_{i=3}^{Length(n)} \chi_{\{0,1\}}(Power(p_i, n)) \right)$$

2. Similarly, we may find the length of n using the *Length* function and apply TapePost to the exponent by using the *Power* function to retrieve the exponent.

Proposition 3.3.2: Important Primitive Recursive Functions

The following are primitive recursive

1. The 4-place function ENTRY (or NEXT) such that

$$ENTRY(j,w,t,n) = \begin{cases} [(t, i+w-1, \mathbf{a}')] & n = [(s,i,\mathbf{a})],\ j \in \{0,1\} \\ & w \in \{0,2\}, i+w-1 \geq 0 \\ & t \geq 1, \text{ where } a'_k = a_k \\ & \text{for } k \neq i \text{ and } a'_i = j. \\ 0 & \text{otherwise} \end{cases}$$

2. For any Turing Machine M with alphabet $\{1\}$, the 1-place function STEP_M such that

$$STEP_M(n) = \begin{cases} [\mathbf{M}(s,i,\mathbf{a})] & \text{if } n = [(s,i,\mathbf{a})] \text{ and} \\ & \mathbf{M}(s,i,\mathbf{a}) \text{ is defined.} \\ 0 & \text{otherwise} \end{cases}$$

3. For any Turing Machine M with alphabet $\{1\}$ the 1-place relation $COMP_M$ where

$$COMP_M(n) \Leftrightarrow n \text{ is the code of a computation } M$$

***Proof* :**

1. Essentially, ENTRY takes in the encoding of a tape position n, say $n = [(s,i,\mathbf{a})]$, and gives the number of the next entry given state t, new position $i + (w-1)$, and tape $\mathbf{a}'$ with $a'_i = j$ and the rest of the tape is identical.
2. Recall that bold $\mathbf{M}$ returns the next tape position. Hence, $STEP_M$ takes in an encoded tape position, and returns the tape position that M would have given.
3. COMP insures that the number n represents a valid computation given a Turing machine M.

Finally, the last important result to show that Turing computable functions are recursive is this next recursive function that requires unbounded mineralization

Proposition 3.3.3: Simulation Function

Let M be a Turing machine with alphabet $\{1\}$. Then the 1-place (partial) function SIM_M is recursive, where

$$SIM_M(n) = [(t, j, \mathbf{b})]$$

if $n = [(1, 0, \mathbf{a})]$ for some input tape a and M eventually halts in position $(t, j, \mathbf{b})$, on input $\mathbf{a}$

Note that $SIM_M(n)$ may be undefined if $n \neq [(1, 0, \mathbf{a})]$ for an input $\mathbf{a}$ or if M does not eventually halt on input $\mathbf{a}$.

Proof **:**
exercise

Hence, we now see that given a Turing machine M and an encoded input take n, we can deduce what the Turing machine will output, given that it halts on the input. The final step we need is simply to insure that encoding and decoding a tape position is indeed recursive:

Lemma 3.3.1: More Primitive Recursive Functions

The following functions are primitive recursive

1. For any fixed $k \geq 1$, $CODE_k(n_1, n_2, ..., n_k) = [(1, 0, 01^{n_1}0...01^{n_k})]$
2. $DECODE(t) = n$ if $t = [(s, i, 01^{n+1})]$ (and anything else otherwise)

Proof **:**
exercise

Theorem 3.3.1: Turing Computable, Then Recursive

Let f be a k-place Turing computable function. Then f is recursive.

Proof **:**
This is now just a combination of everything above.

Hence, since recursive implies Turing computable by theorem 3.2.3, we see that we get

Turing Computable if and only if recursive

3.3.1 Universal Turing Machine and Halting Problem

We can go a little further than what we did above: we can have a recursive function that can simulate *any* Turing machine. Since every recursive function can be computed by a Turing machine, this essentially gives us a *Universal Turing machine*

Definition 3.3.3: Universal Turing Machine (UTM)

A *universal Turing machine* U is a Turing machine such that when given (an appropriate description of) a Turing machine M and an input tape $\mathbf{a}$, it will simulate the computations of M with input $\mathbf{a}$ and gives the output of M on $\mathbf{a}$.

Definition 3.3.4: Encoding Turing Machine

Let M be a Turing machine, hence a partial function as defined in definition 3.1.3. Then if $M(a,b) = (x_1^{(a,b)}, x_2^{(a,b)}, x_3^{(a,b)})$, then:

$$[M] = p_1^{\left(2^{x_1^{(1,0)}} 3^{x_2^{(1,0)}+1} 5^{x_3^{(1,0)}}\right)} p_2^{\left(2^{x_1^{(1,1)}} 3^{x_2^{(1,1)}+1} 5^{x_3^{(1,1)}}\right)} \cdots p_{2n}^{\left(2^{x_1^{(n,1)}} 3^{x_2^{(n,1)}+1} 5^{x_3^{(n,1)}}\right)}$$

where each p_i represents one of the $2n$ possible inputs in the order of $(1,0), (1,1), (2,0), (2,1), ...$, the exponent represents the possible output,and is zero .

Proposition 3.3.4: Generalized STEP And COMP

Let M be a Turing machine. Then the following are primitive recursive

1. $$STEP(m,n) = \begin{cases} [M(s,i,\mathbf{a})] & \text{if } m = [M] \text{ for some machine } M, \\ & \text{if } n = [(s,i,\mathbf{a})] \text{ and } M(s,i,\mathbf{a}) \text{ is defined.} \\ 0 & \text{otherwise} \end{cases}$$

2. $$COMP(m,n) \Leftrightarrow m = [M]$$

   where n is the code of computation of M.

Proof **:**
exercise

Proposition 3.3.5: Universal Turing Machine Simulation

Define SIM in the following way:

$$SIM([M], [(1,0,\mathbf{a})]) = [(t,j,\mathbf{b})]$$

Then it is primitive recursive.

Proof **:**
exercise

As a consequence, there is a Turing machine U which can simulate any Turing machine M. Conversely,

there is a recursive function f which can compute any other recursive functions.

So far, we have been working with the assumption that our machines will halt. Recall that a Tuing machine was was defined to be a partial function, where a lack of state represents the machine halting. Can we determine when a Turing machine will halt? For example, can we always determine when a Turing machine has an infinite loop? This is called the *Halting Problem*, that is:

Given a Turing machine M and an input tape $\mathbf{a}$, is there a method to determine whether or not M eventually halts on input $\mathbf{a}$?

With the definition of a universal Turing machine, we can ask this question more formally:

Definition 3.3.5: The Halting Problem

Is there a Turing Machine T which, for any Turing machine M with alphabet $\{1\}$ and tape $\mathbf{a}$, for M, halts on input

$$\underline{0}1^{[M]+1}01^{[(1,0,\mathbf{a})]+1}$$

with output $\underline{0}11$ if M halts on input $\mathbf{a}$ and with output $\underline{0}1$ if M does not halt on input $\mathbf{a}$?

If you know Church's Thesis, this is equivalent to saying that it is true. We shall not go over in details at the moment, but the answer to the Halting problem as stated above is No.

3.3.2 Recursively enumerable sets

We end off this chapter on computable and recursive functions by taking a slightly different approach to looking at recursive sets. We have defined a set P to be recursive if χ_P is recursive, that is the domain of a recursive function (in particular the characteristic function). What if instead we looked at the image of general recursive function? Will it always be recursive? We explore this here:

Definition 3.3.6: Recursively Enumerable Sets

A subset (i.e. a 1-place relation) $P \subseteq \mathbb{N}$ is *recursively enumerable* (or *r.e.* for short) if there is 1-place recursive function f such that

$$\text{im}(f) = P = \{f(n) \mid n \in \mathbb{N}\}$$

There are many recursively enumerable sets you may come up with: any constant is naturally recursively enumerable, so is the entire $\mathbb{N}$ ($\text{Im}(\pi^1_1)$), and any finite set is naturally recursively enumerable. proper infinite subsets may also be recursively enumerable, for example all even integers are given by:

$$f(n) = Mult(S(S(O(n))), n)$$

However, there do exist recursively enumerable sets that are not recursive:

Proposition 3.3.6: Properties Of Recursively Enumerable Sets

1. If P is a 1-place recursive relation, then P is recursively enumerable (the converse is not always the case)
2. $P \subseteq \mathbb{N}$ is recursive if and only if both P and $\mathbb{N} \setminus P$ are recursively enumerable
3. $P \subseteq \mathbb{N}$ is recursively enumerable if and only if there is a 1-place partial recursive function g such that $P = dom(g) = \{n \mid g(n) \text{ is defined } \}$

***Proof* :**

1. Let P be a 1-place relation. Then, choosing some $p \in P$ we can define $f(n) = n\chi_P(n) + p\chi_{\neg P}(n)$, where evidently $\text{im}(f) = P$ showing that it's recursively enumerable
2. If P is recursive, then we showed in example 3.6 that $\neg P = \mathbb{N} \setminus P$ is recursive, hence they are recursively enumerable. To show the converse, Do the following: If $f(1) = n$, stop, if not then if $g(1) = n$, stop, otherwise if $f(2) = n$, stop, and so on and so forth. Since f and g are both recursive, and $\text{im}(f) \cup \text{im}(g) = \mathbb{N}$, it is guaranteed that at some point we shall find a k such that either $f(k) = n$ or $g(k) = n$. Given this, if H function such that
$$H(t) = \begin{cases} f(t/2) & t = 2l \\ g((t-1)/2) & t = 2l+1 \end{cases}$$
Then
$$\chi_P(n) = div(2, \mu t[|H(t) - n| = 0])$$
is recursive. Note how it was important to use unbounded mineralization for this, would it be possible to prove this without it?
3. exercise: one direction is easy, for the hard direction think of (2)

3.4 Summary

1. **Tape**: an infinite sequence $\mathbf{a} = a_0 a_1 ...$ such that $a_i \in \{0, 1\}$. If $a_i = 0$ for all i, then $\mathbf{a}$ is blank.
2. **Tape Position**: a triple (s, i, a) where
   (a) s will stand for *state* and we will require $s > 0$.
   (b) i will be *index*.
   (c) a will be the tape

   we would visualize a tape position like so:
$$(2, 4, \mathbf{a}) = 2 : 010\underline{1}1001011011$$

3. **Turing Machine**: A partial function: $M : \text{Dom}(M) \to \{0,1\} \times \{-1,1\} \times \{1,...,n\}$ with domain subset of
$$\{1,...,n\} \times \{0,1\}$$
The input represent the current state it's in and what input it's reading on a current tape position, and the output represents what it should override in the current tape position, move either left or right, and go to the next state. Turing machines are usually visualized like so:

M	0	1
1	$1R2$	$0R1$
2	$0L2$	

4. **Computation**: A Turing machine M would, given a tape position, output the information for the successor tape position. Essentially, if (s, i, a) is a tape position and $M(s, a_i) = (b, d, t)$ then the successor tape position is $(t, i + d, a')$ where $a'_i = b$ and $a'_j = a_j$ for the rest of the positions. Then a partial computation with respect to M is a sequence of tape positions p_1... such that each tape position after p_1 are successor tape positions. A partial computation $p_1, p_2, ..., p_k$ is a computation if $p_1 = (1, 0, a)$ (a not necessarily blank) and $M(p_k)$ is undefined (i.e., it halts or terminates)

5. **Computable Function**: Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial function with domain $\text{Dom}(f)$. Then f is *Turing computable* if for each tuple in $\text{Dom}(f)$, when in unary representation
$$\underline{0}1^{n_1+1}01^{n_2+1}...01^{n_k+1}$$
will halt with output tape
$$\underline{0}1^{f(n_1,n_2,...,n_k)+1}$$
Note that not all functions are computable (like the one deduce in the busy beaver competition). Composition of Turing computable functions is Turing computable.

6. **Initial functions and primitive recursion** We know build up to characterizing all computable functions. They will be something called *recursive* which we know build to define. First, we have our building-block functions: O, S, π^k_i, which are all computable. Next, we have primitive recursion which have a base case, and for the inductive case, through the use of initial functions or previously established primitive recursive functions, it has as input all the inputs of your functions, the step of the induction, and the output of the previous step, and will output the next value. Note that a relation is primitive recursive if the characteristic function on the set is primitive recursive.

7. **Examples** see Example 3.5 for important examples as well as lemma 3.2.1 and example 3.6

8. **k-place to 1-place**: Any k-place function or relation can be converted to a 1-place function or relation, and vice-versa. Hence, we may without loss of generality work with 1-place functions/relations

9. **Akerman's Function**: This is a function which is computable but grows faster than any primitive-recursive function can grow, and hence is not primitive recursive

10. **unbounded Minimalization**: given a $k+1$-place function, the unbounded minimalization is the function f where $f(n_1, n_2, ..., n_k) = m$ when $g(n_1, n_2, ..., n_k, m) = 0$. and m is the smallest such function. We require the extra condition that this is always defined, so we either make the domain a partial domain or make guarantee it by saying it's a property of the function that unbounded minimalization is well-defined; such a function is called a *regular function*. Note that bounded minimalization (bounded by a primitive recursive function) is primitive recursive. Unbounded minimalization on regular functions is computable

11. **Recursive function**: finite composition of initial functions, use of primitive recursion, and unbounded minimization is recursive function. If recursive, then computable. We know show that if computable then recursive.

12. **Code of position** Any tape position is given a numerical representation:

$$[(s, i, \mathbf{a})] = 2^2 3^i 5^{a_0} 7^{a_1} 11^{a_2} \cdots p_{n+3}^{a_n}$$

13. **Code of sequence of tape positions**

$$[t_1 t_2 ... t_n] = 2^{[t_1]} 3^{[t_2]} 5^{[t_3]} \cdots p_n^{[t_n]}$$

14. **Code for Turing Machine** see definition 3.3.4

15. **TapePose, TapePosSeq** relations that detect whether a number represents a tape position or a sequence of tape positions. these are both *primitive recursive*

16. **Entry, Step, Comp**: Entry takes in tape position and it's next moves, and gives the corresponding code of the successor tape position. Step takes in a code for a position and output the code of a Turing machine inputting at that position, and $Comp_M(n)$ is true if n is the code of a computation for M.

17. **Sim**: The recursive function Sim_M takes in a code of position $(1, 0, \mathbf{a})$ where $\mathbf{a}$ is blank and gives the result of the computation via Turing Machine M.

18. **Code, Decode**: Code takes in some finite sequence of natural numbers and represents the code of the unitary representation of it, and decode tasks in a code of a output tape position (i.e. after the computation finished) and gives back the actual number.

19. **Computable, then recursive** If f is computable, then it is recursive

20. **Universal Turing Machine (UTM)** a Turing machine that simulates a Turing machine and gives the output of the simulated Turing machine. We can show that a recursive function exists that can simulate any Turing machine, hence there is a Universal Turing machine (and similarly, a recursive function that "has" all recursive functions as part of it)

21. **Recursively Enumerable Sets**: The image of a recursive function is a recursively enumerable sets

22. **Prop of recursively enumerable sets** 1-place recursive, then recursively enumerable. P recursive if and only if P and $\mathbb{N} \setminus P$ recursively enumerable. Recursively enumerable if and only if also domain of partial recursive function.

4

Incompleteness

With the notion of computability established, we can ask a couple of important and foundational questions: given a deductive system, is it possible to find all the theorems that the deductive system may provide, and further more prove these theorems? Perhaps even more fundamentally, can we know for sure that given some choice of axioms for our deductive system, can we be sure now axioms contradict themselves? Due to Gödels work, the answer to both these questions are in general *no*. More particularly, if your system of axioms is strong enough to ask basic logical questions about the natural numbers, then the answer to the above two questions is no! Phrased mathematically, the question becomes:

> *Entscheidungs problem*: Given a reasonable set Σ of formula of a first-order language $\mathcal{L}$ and a formula φ of $\mathcal{L}$, is there an effect method for determining whether or not $\Sigma \vdash \varphi$?

In particular, which roughly asserts that given any set of axioms in a first-order language which are computable and also powerful enough to prove certain facts about arithmetic's, it is possible to formulate statements in the language whose truth is not decided by the axioms! That is, no consistent set of axioms can prove its own consistently! Note that if $\Sigma = \emptyset$, then by the Completeness theorem for any set where $\Gamma \vdash \varphi$ if and only if $\Gamma \models \varphi$, meaning the "empty theory" will not be equivalent to the theory that is strong enough to prove the natural numbers!

In this chapter, we shall first encode the formulas and proofs of a first-order language as numbers and show that the functions and relations involved are recursive. This will make the notion of "computable set of axioms" precise. Next, we will show hat all recursive functions and relations can be defined by first-order formulas in the presence of a fairly minimal set of axioms about elementary number theory, hence linking many axiomatic systems to the axioms of arithmetic's. Finally, by putting recursive functions talking about first-order formulas together with first-order formulas defining recursive functions, we will manufacture a self-referential sentence which asserts its own unprovability, which will lead us to our desired result.

4.1 Deductions are Recursive

The goal of this build-up is to show that any deduction can be computable, i.e. recursive, given a set of premises Δ. After having defined that, we will want to know if every sentence (i.e. every element in a theory) can be shown to be provable or its negation provable, which in a sense makes a theory "complete". Recall that completeness states that $\Gamma \models \sigma$ then $\Gamma \vdash \sigma$, that is completeness was based of off the logical notion of implication. We shall for our purposes borrow the word and use it as a property of a set of sentences rather than a property of a logic:

Definition 4.1.1: Completeness For Set of Sentences

A set of sentences Σ of a first-order langue $\mathcal{L}$ is said to be *complete* if for every sentence τ either $\Sigma \vdash \tau$ or $\Sigma \vdash \neg\tau$.

That is, a set of sentences (or non-logical axioms) is complete if it is sufficient to prove or disprove every sentence of the language in question. If for some sentence σ of $\mathcal{L}$ we cannot deduce whether we can prove (or prove the negation) of σ, then Σ is not complete. An easy source of complete sets comes from chapter 2:

Proposition 4.1.1: Complete iff Maximally Consistent

A consistent set of sentences Σ of a first order-language Λ is complete if and only if the theory

$$\mathrm{Th}(\Sigma) = \{\tau \mid \tau \text{ is a sentence of } \mathcal{L} \text{ and } \Sigma \vdash \tau\}$$

is maximally consistent

Do not confuse $\mathrm{Th}(\Sigma)$ with $\mathrm{Th}(\mathfrak{M}) = \{\tau \mid \tau \text{ is a sentence and } \mathfrak{M} \models \tau\}$.

***Proof* :**
If Σ is complete, then if any sentence $\tau \notin \mathrm{Th}(\Sigma)$, then $\Sigma \nvdash \tau$. Since Σ is complete, it must be that $\Sigma \vdash \neg\tau$, and hence $\neg\tau \in \mathrm{Th}(\Sigma)$. Then adding τ will make $\mathrm{Th}(\Sigma)$ inconsistent, and so $\mathrm{Th}(\Sigma)$ must be maximally consistent.

Conversely, if $\mathrm{Th}(\Sigma)$ is maximally consistent, then if $\tau \notin \Sigma$, $\Sigma \nvdash \tau$. By proposition 2.8.3, $\neg\tau \in \Sigma$, hence $\Sigma \vdash \neg\tau$, completing the proof.

Hence, given a set Σ, we can "detect" when it is complete if $\mathrm{Th}(\Sigma)$ is [maximally] consistent. But how do we link this to computability? For this, we need to encode the logic into something a recursive function can process, similarly in how we encoded tape positions and Turing machines. This is called *Gödel coding*.

Definition 4.1.2: Gödel Code

To each symbol s of $\mathcal{L}_{NT}$, there is a unique positive integer $[s]$called the *Gödel code* of s which is terminated as follows:

1. $[(] = 1$ and $[)] = 2$
2. $[\neg] = 3$ and $[\to] = 4$
3. $[\forall] = 5$
4. $[=] = 6$
5. $[v_k] = k + 12$
6. $[0] = 7$
7. $[S] = 8$
8. $[+] = 9$, $[\cdot] = 10$, $[E] = 11$

Importantly, each positive integers is in bijective correspondence with the symbols of $\mathcal{L}_{NT}$.

Definition 4.1.3: Gödel Codes Of Sequences

Let $s_1 s_2 \cdots s_k$ be a sequence of symbols of $\mathcal{L}_{NT}$. Then the *Gödel code* of this sequence is:

$$[s_1 \cdots s_k] = p_1^{[s_1]} \cdots p_k^{[s_k]}$$

where p_i is the ith prime number. If we have a sequence of sequences, we go through them recursively.

Example 4.1: Gödel Coding

1. Take $\forall v_1 = \cdot v_1 S0 v_1$ (which is $\forall v_1 \; v_1 \cdot S0 = v_1$). Then: $[\forall v_1 = \cdot v_1 S0 v_1]$ would be:

   $$2^{[\forall]} 3^{[v_1]} 5^{[=]} 7^{[\cdot]} 11^{[v_1]} 13^{[S]} 17^{[0]} 19^{[v_1]} = 2^5 3^{13} 5^6 7^{10} 11^{13} 13^8 17^7 19^{13}$$

   which equals to

   $$109425289274918632559342112641443058962750733001979829025245569500000$$

2. If we take the sequence of formulas

   $$\begin{aligned} &= 00 \\ &(= 00 \to = S0S0) \\ &= S0S0 \end{aligned}$$

   which translates to "Given $0 = 0$, and $0 = 0 \to S0 = S0$, then $S0 = S0$", what is the given Gödel code?

Note that an integer n may simultaneously be the Gödel code of a symbol, a sequence of symbols, and a sequence of a sequences of symbols. It will be up to context to determine which one.

Next, we will need a way of recognizing Gödel codes and verify whether it encodes a sentence or a logical axiom:

Proposition 4.1.2: Relations To Manipulate Gödel Codes

The following functions are primitive recursive

1. $Term(n) \Leftrightarrow n = [t]$ for some term t of $\mathcal{L}_{NT}$
2. $Formula(n) \Leftrightarrow n = [\varphi]$ for some formula φ of $\mathcal{L}_{NT}$
3. $Sentence(n) \Leftrightarrow n = [\sigma]$ for some sentence σ of $\mathcal{L}_{NT}$
4. $Logical(n) \Leftrightarrow n = [\gamma]$ for some logical axiom γ of $\mathcal{L}_{NT}$

***Proof* :**
exercise. Be careful on verifying *Sentence* since you need to check there are no free variables.

With these functions, we can develop relation and functions to handle deductions of $\mathcal{L}_{NT}$. We first can make the notion of computable set of formulas precise:

Definition 4.1.4: Recursive Set

A set of formulas Δ of $\mathcal{L}_{NT}$ is *recursive* is the set of Gödel codes of the formula

$$[\Delta] = \{[\delta] \mid \delta \in \Delta\}$$

is a recursive subset of $\mathbb{N}$ (i.e. a recursive 1-place relation). Similarly, Δ is said to be *recursively enumerable* if $[\Delta]$ is recursively enumerable

Proposition 4.1.3: Relations To Recursive Set

Let Δ be a recursive set of sentences $\mathcal{L}_{NT}$. Then the following relations are recursive:

1. $Premise_{\Delta}(n) \Leftrightarrow n = [\beta]$ for some formula β of $\mathcal{L}_{NT}$ which is either a logical axiom or in Δ
2. $Formula(n) \Leftrightarrow n = [\varphi_1\varphi_2...\varphi_k]$ for some sequence $\varphi_1, \varphi_2, ..., \varphi_k$ of formulas of $\mathcal{L}_{NT}$
3. $Inference(n, i, j) \Leftrightarrow n = [\varphi_1\varphi_2...\varphi_k]$ for some sequence $\varphi_1, \varphi_2, ..., \varphi_k$ of formulas of $\mathcal{L}_{NT}$, $1 \leq i, j \leq k$, and φ_k follows from φ_i and φ_j by Modus Ponens
4. $Deduction_{\Delta}(n) \Leftrightarrow n = [\varphi_1\varphi_2...\varphi_k]$ for a deduction $\varphi_1, \varphi_2, ..., \varphi_k$ from Δ in $\mathcal{L}_{NT}$
5. $Conclusion_{\Delta}(n, m) \Leftrightarrow n = [\varphi_1\varphi_2...\varphi_k]$ for a deduction$\varphi_1, \varphi_2, ..., \varphi_k$ from Δ in $\mathcal{L}_{NT}$ and $m = [\varphi_k]$

Proof :
exercise

As an exercise, if $[\Delta]$ is primitive recursive, which of the above are also primitive recursive? With this setup, we may now make the connection between computability and completeness:

Theorem 4.1.1: Computability And Completeness

Let Δ be a recursive set of sentences of $\mathcal{L}_{NT}$. Then $[Th(\Delta)]$ is

1. recursively enumerable, and
2. recursive if and only if Δ is complete

Proof :

1. First, recall that
$$[\varphi_1\varphi_2...\varphi_k] = 2^{[\varphi_1]}3^{[\varphi_2]}\cdots p_k^{[\varphi_k]}$$
hence, each prime number stores the information about each $[\varphi_i]$. To insure n is of the above form, we may use
$$Deduction_\Delta(n)$$
to check that n is indeed a deduction. We next need to find m so that $Conclusion_\Delta(n,m)$ is true, that is we need $[\varphi_k]$. Take
$$Power(Pick(Element(n, Length(n))), n) \tag{4.1}$$
where
$$Pick(p_i^k) = \sum_{n=0}^{p_i^k} Mult(Prime(n), Div(Prime(n), p_i^k)) = p_i$$
decomposing equation (4.1), it will first select the last prime power of n (i.e. the prime storing the information about φ_k), then $Pick$ will extract the prime as we've described in 2b, then Power will give us the exponent of $Element(n, length(n))$, namely $[\varphi_k]$. Hence:
$$f(n) = Deduction_\Delta(n) \cdot Power(Pick(Element(n, Length(n))), n)$$
since these are all recursive functions, f is recursive. Hence to finish the formalisms we get:
$$f(n) = Deduction_\Delta(\pi_1^1) \cdot Power(Pick(Element(n, Length(\pi_1^1))), \pi_1^1) \circ (n)$$
The domain of f can either be all of $\mathbb{N}$, in which case if a particular n does not represent a deduction it will map to 0, or we can have a partial function that is only defined on $(\chi_{Deduction_\Delta})^{-1}(\{1\})$. The image is:
$$\text{Im}(f) = \{m \mid m = [\varphi_k],\ \Delta \vdash \varphi_k\} = [Th(\Delta)]$$

2. Let's first say that $[Th(\Delta)]$ is recursive so that $\chi_{\{[Th(\Delta)]\}}$ is recursive. For the following argument, we may make this function a partial function on the set of sentences, which is

recursive since $Sentence(n)$ is recursive. We want to show that for any sentence τ, either $\Delta \vdash \tau$ or $\Delta \vdash \neg\tau$. If $\chi_{[Th(\Delta)]}(n) = 1$, then $\Delta \vdash \tau$ and if $\chi_{[Th(\Delta)]}(n) = 0$, then $\Delta \nvdash \tau$.

Now, if Δ is consistent, then $\Delta \vdash \tau$ implies $\Delta \nvdash \neg\tau$ (or else Δ is inconsistent). If $\Delta \vdash \neg\tau$, then $\Delta \nvdash \tau$ (for the same reason). If Δ is inconsistent, then $Th(\Delta)$ is every possible sentence since an inconsistent set can prove sentence. Hence, Δ is complete.

Conversely, let's say Δ was complete so that $\Delta \vdash \tau$ or $\Delta \vdash \neg\tau$. We want to show that $\chi_{[Th(\Delta)]}$ is recursive. If $Sentence(n)$ is false, then certainly $\chi_{[Th(\Delta)]} = 0$, so we need to make check for when $Sentence(n)$ is true. Next, by part 1, we know that $[Th(\Delta)]$ is recursively enumerable, so there exists a recursive function f such that $\text{Im}(f) = [Th(\Delta)]$. Since Δ is complete, either τ or $\neg\tau$ is in $Th(\Delta)$. Each of these are represented by finite numbers in $[Th(\Delta)]$, hence by unbounded minimalization there must exist an n such that

$$f(n) = [\tau] \qquad \text{or} \qquad f(n) = [\neg\tau]$$

If $m = [\tau]$, then let $\mu m[f]$ be the unbounded minimization that either equals $m = [\tau]$ or $m' = [\neg\tau]$. With this, define

$$\chi_{[Th(\Delta)]}(m) = Sentence(m) \cdot \chi_{\{0\}}(absDiff(\mu m[f], m))$$

Then this function is recursive, being a composition of recursive function, and hence $[Th(\Delta)]$ is recursive, completing the proof.

Note that it follows that if Δ is not complete, then $[Th(\Delta)]$ is an example of a recursively enumerable set that is not a recursive set.

4.2 Recursive Functions are Representable in $\mathcal{A}$

Let's now take a particular Σ and study whether it is complete. In particular, we shall make a list of [non-logical] axioms for the natural numbers (labelled $\mathcal{A}$) and verify the completeness of $\mathcal{A}$, or slightly broader the completeness of Σ if $\Sigma \vdash \mathcal{A}$. The reason for this broadening is because a lot of systems let us deduce the axioms of natural numbers (a lot of mathematics is based of of some property of natural numbers), and so this let's us make an interesting statement about many sets of axioms.

Recall the following

Definition 4.2.1: Number Theory Language

Let $\mathcal{L}_{NT}$ be the language generated by $\{0, S, +, \cdot, E, <\}$, where 0 is a constant, S is a unary function (the "successor function"), $+$, $\cdot$, and E are binary function (E for exponential) and $<$ is a binary relation. Then we usually call $\mathcal{L}_{NT}$ the language of *number theory*

Naturally, the standard structure for the language of number theory is

$$\mathfrak{N} = (\mathbb{N}, 0, S, +, \cdot, E, <)$$

From what we've built up, we can use the natural numbers and recursive functions to code and manipulate formulas of $\mathcal{L}_{NT}$. We now need the complementary results that let us use terms and

formulas of $\mathcal{L}_{NT}$ to represent and manipulate natural numbers and recursive functions. What we will do is create a set of non-logical axioms in $\mathcal{L}_{NT}$ which prove enough about the operations of successor, addition, multiplication, and exponentiation to let us define all recursive functions using formula of $\mathcal{L}_{NT}$.

Definition 4.2.2: Non-Logical Axioms Of Number Theory ($\mathcal{A}$)

Let $\mathcal{A}$ be the following set of sentences of $\mathcal{L}_{NT}$, written out in official form:

1. $\forall v_0(\neg = Sv_0 0)$
2. $\forall v_0((\neg = v_0) \to (\neg\forall v_1(Sv_1 v_0)))$
3. $\forall v_0 \forall v_1(= Sv_0 Sv_1 \to = v_0 v_1)$
4. $\forall v_0 = +v_0 0 v_0$
5. $\forall v_0 \forall v_1 = +v_0 Sv_1 S + v_0 v_1$
6. $\forall v_0 = \cdot v_0 00$
7. $\forall v_0 \forall v_1 = \cdot v_0 Sv_1 + \cdot v_0 v_1 v_0$
8. $\forall v_0 = Ev_0 S0$
9. $\forall v_0 \forall v_1 Ev_0 Sv_1 \cdot Ev_0 v_1 v_0$

In colloquial terms, we can re-think these as follows;

1. For all n, $n + 1 \neq 0$
2. for all n, $n \neq 0$ there is a k such that $k + 1 = n$
3. for all n, and k, $n + 1 = k + 1$ implies $n = k$
4. for all n, $n + 0 = n$
5. for all n, and k, $n + (k + 1) = (n + k) + 1$
6. for all n, $n \cdot 0 = 0$
7. for all n, and k, $n \cdot (k + 1) = (n \cdot k) + n$
8. for all n, $n^0 = 1$
9. for all n, and k, $n^{k+1} = (n^k) \cdot n$

Proposition 4.2.1: Natural Number Axioms Are True

Let $\mathfrak{N} = (\mathbb{N}, 0, S, +, \cdot, E)$ is the structure consisting of the natural numbers and the usual functions. Then

$$\mathfrak{N} \models \mathcal{A}$$

***Proof* :**
exercise

We might want that from $\mathcal{A}$ we can prove all sentences of first-order arithmetic are true in $\mathfrak{N}$, however this is not the case. For example, $\mathcal{A}$ is not enough to insure that inductions works, that is for every formula φ with at most one free variables x, if φ_0^x and $\forall y(\varphi_y^x \to \varphi_{Sy}^x)$ holds, then so does $\forall x\varphi$ (we shall not prove this here). On the other hand, neither $\mathcal{L}_{NT}$ nor $\mathcal{A}$ are quite as minimal as we can make them. For example, with a considerable amount of extra effort, we can eliminate E and define $\cdot$ in terms of $+$. However, redundancy is chosen since the goal is not to be as minimal as possible, but to find the provability of statements in $\mathfrak{N}$, for which it would be better if we can use some stronger axioms to simplify our results.

Now, we can move on to representing functions and relations. To not clog up notations, let S^m0 represent concatenating m many S symbols.

Lemma 4.2.1: Repeated Successor Function

For every $m \in \mathbb{N}$ and every assignment function s for $\mathfrak{N}$,

$$\mathbf{s}(S^m0) = m$$

***Proof* :**
$0^{\mathfrak{N}}$ is assocaited to 0, and going through the definiting gives:

$$\underbrace{1+1+1+\cdots+1}_{m \text{ times}} = m$$

The next abbreviation we shall do will be to help represent function input. In particular, if we have $v_1, v_2, ..., v_k$ free variables and $m_1, m_2, ..., m_k$ natural numbers, then

$$\varphi(S^{m_1}0...., S^{m_k}0) :\equiv \varphi^{v_1,...,v_k}_{S^{m_1}0,...,S^{m_k}0}$$

Notice that each $S^{m_i}0$ have no variables, and hence they are substitutable

Definition 4.2.3: Representable Function

Let Σ be a set of sentences of $\mathcal{L}_{NT}$. Then a k-place function f is said to be *representable* in $\mathrm{Th}(\Sigma) = \{\tau \mid \Sigma \vdash \tau\}$ if there is a formula φ of $\mathcal{L}_{NT}$ with at most $v_1, v_2, ..., v_k$ and v_{k+1} as free variables such that

$$\begin{aligned} f(n_1, n_2, ..., n_k) = m &\Leftrightarrow \varphi(S^{n_1}0, ..., S^{n_k}0, S^m0) \in \mathrm{Th}(\Sigma) \\ &\Leftrightarrow \Sigma \vdash \varphi(S^{n_1}0, ..., S^{n_k}0, S^m0) \end{aligned}$$

for all $n_1, n_2, ..., n_k$ and m in $\mathbb{N}$. The formula φ is said to *represent* f in $\mathrm{Th}(\Sigma)$.

Usually, we'll have $\Sigma = \mathcal{A}$ or Σ where $\Sigma \vdash \mathcal{A}$.

Example 4.2: Representable Functions

1. I claim that the constant function c_3^1, $c_3^1(n) = 3$, is representable in $\mathrm{Th}(\mathcal{A})$, namely the formulas $v_2 = S_0^3$ represents it. We need to show

$$\begin{gathered} c_3^1(n) = m \\ \Leftrightarrow \\ \mathcal{A} \vdash [v_2 = S^3 0](S^n 0, S^m 0) \\ \Leftrightarrow \\ \mathcal{A} \vdash S^m 0 = S^3 0 \end{gathered}$$

   for all $n, m \in \mathbb{N}$. To show $\Rightarrow$, by definition $c_3^1(n) = 3 = m$. Hence, we want to show that $\mathcal{A} \vdash S^3 0 = S^3 0$. The deduction is straightforward:

(1) $\forall x\ x = x \to S^3 0 = S^3 0$	$A4$
(2) $\forall x\ x = x$	$A8$
(3) $S^3 0 = S^3 0$	$1, 2\ MP$

   Hence, if $c_1^3(n) = m$,then $\mathcal{A} \vdash S^m 0 = S^3 0$. Conversely, suppose $\mathcal{A} \vdash S^m 0 = S^3 0$. Since $\mathfrak{N} \models \mathcal{A}$, $\mathfrak{M} \models S^m 0 = S^3 0$. Then by lemma 4.2.1 $m = 3$, so $c_3^1(n) = m$, completing the other direction.

2. Show that the zero function, $O(n)$, is representable

3. Let $S(n)$ be the successor function, and let $\varphi :\equiv v_2 = S(v_1)$. Then we need to show:

$$\begin{gathered} S(n) = m \\ \Leftrightarrow \\ \mathcal{A} \vdash [v_2 = S v_1](S^n 0, S^m 0) \\ \Leftrightarrow \\ \mathcal{A} \vdash S^m 0 = S^{n+1} 0 \end{gathered}$$

   Since $m = S(n) = n + 1$, we get $\mathcal{A} \vdash S^{n+1} = S^{n+1}$, which by our above proof we know is deducible. Conversely, if $\mathcal{A} \vdash S^m = S^{n+1}$, then since $\mathfrak{M} \models \mathcal{A}$ we get $\mathfrak{M} \models S^m = S^{n+1}$, which by lemma 4.2.1 we know implies $m = n + 1$, which implies $S(n) = n + 1$, as we sought to show.

4. Show that the projection function, π_i^k, is representable

Definition 4.2.4: Representable Relation

Let $P \subseteq \mathbb{N}^k$ be a k-place relation. Then P is *representable* in $\text{Th}(\Sigma)$ is there is a formula ψ of $\mathcal{L}_{NT}$ with at most $v_1, v_2, ..., v_k$ free variables such that

$$\begin{aligned} P(n_1, n_2, ..., n_k) &\Leftrightarrow \varphi(S^{n_1}0, ..., S^{n_k}0) \in \text{Th}(\Sigma) \\ &\Leftrightarrow \Sigma \vdash \varphi(S^{n_1}0, ..., S^{n_k}0) \end{aligned}$$

for all $n_1, n_2, ..., n_k \in \mathbb{N}$. The formula ψ is said to *represent* $\text{Th}(\Sigma)$.

Example 4.3: Representable Relation

1. Take the 1-place relation $P = \{3\}$. Show that the formula $v_1 = S^3 0$ represents this relation (i.e. this set). Note that $v_2 = S^3 0$ does not represent this set, and $v_1 = S^3 0$ does not represent the function in example 4.2(1)
2. Let $P = 2\mathbb{N}$. Show that the representative of this relation is $\exists v_2, v_1 = (S^2 0 \cdot (v_2 \cdot S0))$, which more casually would look like $\exists k,\ n = 2k$.

Proposition 4.2.2: Relating Representable Functions and Relations

Let f be a k-place function. Then f is a representable in $\text{Th}(\mathcal{A})$ if and only if the $k+1$ place relation P_f defined by

$$P_f(n_1, n_2, ..., n_k, n_{k+1}) = f(n_1, n_2, ..., n_k) = n_{k+1}$$

Conversely, a relation $P \subseteq \mathbb{N}^k$ is representable in $\text{Th}(\mathcal{A})$ if and only if its characteristic function χ_P is representable in $\text{Th}(\mathcal{A})$.

***Proof* :**
exercise, think of what formula you would use in each case.

We may ask what functions and relations are representable. Importantly for us, recursive functions (equivalently, computable function) representable. We shall spend the rest of this section showing this inclusion. We already showed the initial functions are representable in example 4.2. We need to show finite composition, unbounded minimalization, and primitive recursion are all representable. Starting with finite composition:

Proposition 4.2.3: Composition Of Representable Is Representable

Let $g_1, g_2, ..., g_m$ be k-place functions and let h an m-place function, where all these functions are representation in $\text{Th}(\mathcal{A})$. Then $f = h \circ (g_1, g_2, ..., g_m)$ is a k-place representable function in $\text{Th}(\mathcal{A})$.

Proof :
It suffices to show it in the case of $h = g \circ f$. Let α represent f and β represent g so that

$$f(n) = m \Leftrightarrow \Sigma \vdash \alpha(S^n 0, S^m 0) \qquad g(m) = k \Leftrightarrow \Sigma \vdash \beta(S^m 0, S^k 0)$$

Then consider the formula

$$\exists v_3 \beta^{v_1, v_2}(v_3, S^k) \wedge \alpha^{v_1, v_2}(S^n 0, v_3)$$

it should not be too hard to see that this formula represents $g \circ f$

Proposition 4.2.4: Representable Functions And Unbounded Minimalization

Let g be a $k+1$-place regular function which is representable in Th($\mathcal{A}$). Then the unbounded minimalization of g is a k-place representable function in Th($\mathcal{A}$)

Proof :
think about what the formula α should be given φ represents g. Remember that $f(n_1, n_2, ..., n_k) = m$ if and only if m is least so that $g(n_1, n_2, ..., n_k, m) = 0$.

What's left to show is that primitive recursive functions are representable. The main problem is that it is not immediately obvious how a formula defining a function gets the previous value of the function. We will essentially encode this information by having a "history" function. To define this function, we need to show the functions in lemma 4.2.2 are representable[1]:

Lemma 4.2.2: Important Representable Functions and Relations

The following functions and relations are representable in Th($\mathcal{A}$):

1. $Div(n, m) \Leftrightarrow n \big| m$
2. $IsPrime(n) \Leftrightarrow n$ is prime
3. $Prime(k) = p_k$, where $p_0 = 1$ and p_k is the kth prime if $k \geq 1$
4. $Power(n, m) = k$ where $k \geq 0$ and is maximal such that $n^k \big| m$
5. $Length(n) = \ell$ where ℓ is maximal such that $p_\ell \big| n$
6. $Element(n, i) = n_i$ where $n = p_1^{n_1} ... p_k^{n_k}$ (and $n_i = 0$ if ik)

Proof :
these are good exercises

[1] All of these functions were shown to be primitive recursive using primitive recursion, hence we cannot say they are just a finite composition of initial functions

Proposition 4.2.5: History Functions

Let f be a k-place function that is representable in $\mathrm{Th}(f)$. Then

$$F(n_1, n_2, ..., n_k, m) = p_1^{f(n_1,n_2,...,n_k,0)} ... p_k^{f(n_1,n_2,...,n_k,m)} = \prod_i^m p_i^{f(n_1,n_2,...,n_k,i)}$$

is representable in $\mathrm{Th}(\mathcal{A})$

Proof **:**
good exercise

Proposition 4.2.6: Primitive Recursive, Then Representable

Let g be a k-place function and h a $k+2$ place function where both a Representable in $\mathrm{Th}(\mathcal{A})$. Then the $k+1$-place function f defined by primitive recursion from g and h is also representable in $\mathrm{Th}(\mathcal{A})$.

Proof **:**
use the above proposition to keep track of the history.

Theorem 4.2.1: Recursive, Then Representable

Let f be a recursive function. Then f is representable in $\mathrm{Th}(\mathcal{A})$

Proof **:**
combine the previous results.

Hence, there are formulas of $\mathcal{L}_{NT}$ that represent each of the functions which encode assertions about terms, formulas, deductions, and so forth. This will be crucial in the Incompleteness Theorem, since we will show there is a formula which will code its own unprovability.

We end this section by giving a couple more important results about representability in relation to consistency:

Proposition 4.2.7: Representability And Consistency

1. Let Σ be a set of sentences of $\mathcal{L}_{NT}$ and f a k-place representable function in $\text{Th}(\Sigma)$. Then Σ is consistent
2. Let Σ and Γ be consistent sets of sentences of $\mathcal{L}_{NT}$ and $\Sigma \vdash \Gamma$. Then every function and relation which is representable in $\text{Th}(\Gamma)$ is representable in $\text{Th}(\Sigma)$
3. For any consistent set of sentences Σ such that $\Sigma \vdash \mathcal{A}$, any function and relation representable in $\text{Th}(\mathcal{A})$ is also representable in $\text{Th}(\Sigma)$.

***Proof* :**

1. Proving the contrapositive, recall that if Σ is inconsistent, then Σ can prove anything. Let's say f is a function where $f(n) \neq m$. Then since Σ is inconsistent, $\Sigma\varphi(S^n, S^m)$, however $f(n) \neq m$.
2. Let's say f is representable in $\text{Th}(\Gamma)$ with representative φ. Since $\Sigma \vdash \Gamma$ and $\Gamma \vdash \varphi(S^{n_1}0, ..., S^{n_k}0, S^m0)$ whenever $f(n_1, n_2, ..., n_k) = m$, by transitivity so does Σ
3. let $\Gamma = \mathcal{A}$ in the above.

4.3 Incompleteness Theorems

We have shown that we can use recursive functions to manipulate coded formulas of $\mathcal{L}_{NT}$, and have showed that we can represent recursive function using formulas in $\mathcal{L}_{NT}$. This leaves us open to "self-referencing", in particular we can use formulas of $\mathcal{L}_{NT}$ to refer and manipulate codes of formulas of $\mathcal{L}_{NT}$.

The two key results we'll be proving are Gödels two incompleteness theorems. It essentially states that if Σ is a consistent and computable (i.e. recursive) set of sentences of $\mathcal{L}_{NT}$ that is large enough so that $\Sigma \vdash \mathcal{A}$, then Σ is *not* complete, that is it cannot be that for any sentence τ of $\mathcal{L}_{NT}$ then either $\Sigma \vdash \tau$ or $\Sigma \vdash \neg\tau$. We then move onto the second incompleteness theorem which further asserts that a consistent computable (i.e. recursive) set of sentences Σ of $\mathcal{L}_{NT}$ *cannot* prove its own consistency!

To prove the first incompleteness theorem, we first need a couple of lemmas:

Lemma 4.3.1: Incompleteness Theorem Lemmas I

1. The function
$$Sub(n,k) = \begin{cases} [\varphi(S^k 0)] & \text{if } n = [\varphi] \text{ fora formula} \\ & \varphi \text{ of } \mathcal{L}_{NT} \text{ with} \\ & \text{at most } v_1 \text{ free} \\ 0 & \text{otherwise} \end{cases}$$
is recursive, and hence representable in $\text{Th}(\mathcal{A})$
2. $\mathcal{A}$ is a recursive set of sentences in $\mathcal{L}_{NT}$
3. (Fixed-Point lemma) Let φ be a formula of $\mathcal{L}_{NT}$ with only v_1 as a free variable. Then there is a sentence σ of $\mathcal{L}_{NT}$ such that
$$\mathcal{A} \vdash \sigma \leftrightarrow \varphi(S^{[\sigma]}0)$$

Note that for the first lemma, σ must be different from the sentence $\varphi(S^{[\varphi]}0)$, since there is no way to find a formula φ with one free variable and an integer k such that $[\varphi(S^k 0)] = k$. Also, the fixed point lemma is sometimes called the *Diagonalization Lemma* due to parallels between the proof argument and Cantor's Diagonalization theorem proving there are uncountably many real numbers.

***Proof* :**

1. Recall that a formula has Gödel code $k + 12$. Hence, φ if there exists p_i's such that the exponent is greater than 13, p_j^{k+13} then we must verify that its bounded, that is in an earlier element $p_{i_1}^5 p_{i_2}^{k+13}$, $i_1, i_2 < j$ and $i_2 = i_1 + 1$.
2. exercise
3. We first need to find a representation of Sub. Since Sub is recursive, it is representable in $\text{Th}(\mathcal{A})$, say ψ represents it:
$$Sub(n,k) = [\varphi(S^k 0)] \Leftrightarrow \mathcal{A} \vdash \psi(S^n 0, S^k 0, S^{[\varphi(S^k 0)]})$$
Note that nothing is stopping us from doing $Sub([\varphi],[\varphi])$, or equivalently if $n = [\varphi]$ then $\psi(S^n, S^n, S^m 0)$ where m is the output of the above function. Let $n = [\varphi]$ for the rest of this proof. Now, given φ, we can define the following formula:
$$\alpha :\equiv \exists v_2[\varphi(v_2) \wedge \psi(v_1, v_1, v_2)]$$
Remember that $\varphi(v_2) :\equiv \varphi_{v_2}^{v_1}$ and $\psi(v_1, v_1, v_2) :\equiv \psi_{v_1,v_2,v_2}^{v_1,v_2,v_3}$. Note that α only has one free variable, namely v_1. Let $k = [\alpha]$. Then consider
$$\alpha(S^k 0) :\equiv \exists v_2[\varphi(v_2) \wedge \psi(S^k 0, S^k 0, v_2)]$$
If we take $m = [\alpha(S^k 0)]$, then
$$Sub(k,k) = m$$
and furthermore:
$$\mathcal{A} \vdash \forall v_2[\psi(S^k 0, S^k 0, v_2) \leftrightarrow y = m]$$

which, to make more clear what was written we have:

$$\mathcal{A} \vdash \forall v_2[\psi(S^k 0, S^k 0, v_2) \leftrightarrow y = [\alpha(S^k)]]$$

Now, using the definition of $\leftrightarrow$ and simplifying notation, we get:

$$\mathcal{A} \vdash \alpha(S^k 0) \leftrightarrow \psi(S^{[\alpha(S^k 0)]})$$

The fixed point lemma can be thought of as the statement "This statement is false", which is the key paradox we will use to prove the incompleteness theorem[2]

Theorem 4.3.1: Gödels First Incompleteness Theorem I

Let Σ be a consistent recursive set of sentences of $\mathcal{L}_{NT}$ such that $\Sigma \vdash \mathcal{A}$. Then Σ is *not* complete

***Proof* :**
For the sake of contradiction, assume Σ is complete so that $\text{Th}(\Sigma)$ is recursive. Then it must be representable in $\text{Th}(\mathcal{A})$, say φ represents it. Then by the fixed point lemma, we may find a formual σ of $\mathcal{L}_{NT}$ such that $\mathcal{A} \vdash \sigma \leftrightarrow \neg\varphi(S^{[\sigma]}0)$ (note the negation). If $\sigma \in \text{Th}(\Sigma)$, then $\Sigma \vdash \sigma$, and $\mathcal{A} \vdash \sigma$, implying $\Sigma \vdash \neg\varphi(S^{[\sigma]}0)$, which means that $[\sigma]$ is not in the set of codes for $\text{Th}(\Sigma)$, implying $\sigma \notin \text{Th}(\Sigma)$. On the other hand, if $\sigma \notin \text{Th}(\Sigma)$, then $\Sigma \vdash \neg\sigma$ since Σ is complete, hence $\Sigma \vdash \neg\neg\varphi(S^{[\sigma]}0)$, which we know lets us deduce $\Sigma \vdash \varphi(S^{[\sigma]}0)$ which implies $\sigma \in \text{Th}(\Sigma)$. But it cannot be that both σ is both in and not in $\text{Th}(\Sigma)$, a contradiction. Hence Σ cannot be complete, as we sought to show.

This theorem in particular states that any consistent set of sentence which proves at least as much about the natural numbers as $\mathcal{A}$ does cannot be both complete and recursive (i.e. computable)!

Corollary 4.3.1: Consequences Of First Incompleteness Theorem

1. Let Γ be a complete set of sentences of $\mathcal{L}_{NT}$ such that $\Gamma \cup \mathcal{A}$ is consistent. Then Γ is *not* recursive
2. Let Δ be a recursive set of sentences such that $\Delta \cup \mathcal{A}$ is consistent. Then Δ is *not* complete
3. The theory of $\mathfrak{N}$

   $$\text{Th}(\mathfrak{N}) = \{\sigma \mid \sigma \text{ is a sentence of } \mathcal{L}_{NT} \text{ and } \mathfrak{N} \models \sigma \}$$

   is *not* recursive

[2]There are other paradoxes that have similar properties (for example, the barber paradox), and each paradox leads to a different proof of the incompleteness theorem

***Proof* :**
The first two come from the fact that there are no consistent complete recursive set of sentences such that $\Sigma \vdash \mathcal{A}$. The third has the same reasoning: we know that $\text{Th}(\mathfrak{N})$ is maximally consistent by proposition 2.9.4. For completeness,pick any set of generating sentences, that is a subset $N \subseteq \text{Th}(\mathfrak{N})$ such that $\text{Th}(N) = \text{Th}(\mathfrak{N})$ (note how the two sets are defined differently, but we use the Completeness theorem from first-order logic to equate the two). Hence, by proposition 4.1.1, N is complete. Furthermore, we trivially have $N \vdash \mathcal{A}$ (since $\text{Th}(\mathfrak{N}) \supseteq \text{Th}(\mathcal{A})$), and hence we may apply Gödel's Incompleteness Theorem.

We were working over $\mathcal{L}_{NT}$, however nothing particular about this particular language was important for the result; it works perfectly well for any first order language and recursive set of axioms where we can encode and prove enough facts about arithmetic. Notably, it can be done whenever the language and axioms are powerful enough to define the natural numbers and prove some simple facts about them (a popular example would be Zermelo-Fraenkel set Theory).

We now move onto the second incompleteness theorem, which as mentioned before asserts that a consistent recursive se of sentences Σ for $\mathcal{L}_{NT}$ cannot prove its own consistency. We first require the ability to express a notion of "Σ is consistent" in $\mathcal{L}_{NT}$

Lemma 4.3.2: Expressing Consistency

Let Σ be a recursive set of sentences of $\mathcal{L}_{NT}$. Then there is a sentence of $\mathcal{L}_{NT}$, denoted $\text{Con}(\Sigma)$, such that Σ is consistent if and only if $\mathcal{A} \vdash \text{Conn}(\Sigma)$

***Proof* :**
hint: consider $\text{Con}(\Sigma)$ to be $\forall v_1 \neg \mathit{Conclusion}(v_1, [(\neg(\alpha \to \alpha))])$, i.e. $\neg\exists v_1,\ \mathit{Conclusion}(v_1, [(\neg(\alpha \to \alpha))])$

Theorem 4.3.2: Gödels Second Incompleteness Theorem II

Let Σ be a consistent recursive set of sentences of $\mathcal{L}_{NT}$ such that $\Sigma \vdash \mathcal{A}$. Then $\Sigma \nvdash \text{Con}(\Sigma)$

***Proof* :**
exercise

Just like for the first incompleteness theorem, the second one works for any recursive set of sentences in a first order language which allows one to encode and prove enough facts about arithmetic. One "pathological" consequence of the Second Incompleteness Theorem is that only an inconsistent set of axioms can prove its own consistency!!

We end off this section with a couple of implications of incompleteness. First, recall that a sentence σ is true in $\mathfrak{N}$ if $\mathfrak{N} \models \sigma$. We shall show that a true sentence in $\mathfrak{N}$ is not "definable" in $\mathfrak{N}$. We first require to define our term:

Definition 4.3.1: Definable

Let P be a k-place relation in $\mathfrak{N}$. Then P is said to be *definable* if there is a formula φ of $\mathcal{L}_{NT}$ wit hat most $v_1, v_2, ..., v_k$ as free variables such that

$$P(n_1, n_2, ..., n_k) \Leftrightarrow \mathfrak{N} \models \varphi[s(v_1|n_1)...(v_k|n_k)]$$

for every assignment s of $\mathfrak{N}$. The formula φ will be said to *define* P in $\mathfrak{N}$.

A similar definition can be me for functions. Notice too how similar being definable is to being representable. The following proposition makes the relation more precise

Proposition 4.3.1: Representable And Definable

Let P be a k-place relation which is representable in $\text{Th}(\mathcal{A})$. Then P is *definable* in $\mathfrak{N}$

Proof **:**
exercise

(note to self: check if converse is true, I believe it is)

From this, we see that the question of whether truth is definable in $\mathfrak{N}$ comes down to whether the set of Gödel codes of sentences $\mathcal{L}_{NT}$ which are true in $\mathfrak{N}$,

$$[\text{Th}(\mathfrak{N})] = \{[\sigma] \mid \sigma \text{ is a sentence of } \mathcal{L}_{NT} \text{ and } \mathfrak{N} \models \sigma\}$$

are definable. The following tells us this is not the case

Theorem 4.3.3: Tarski'S Undefinability Theorem

The set $[\text{Th}(\mathfrak{N})]$ is *not* definable in $\mathfrak{N}$

Proof **:**
see this here.

4.4 Summary

Overall, the first incompleteness theorems showed there is no real procedure that will find and prove all theorems. The second incompleteness theorem shows that we can never be certain that any reasonable set of axioms (i.e. encode at least the arithmetic of the natural numbers) is consistent, unless we take a larger set of axioms on faith. Hence, besides simply having faith, we can never be completely sure the theorems proved in mathematics are really true.

1. **Language of Number Theory** We will be sticking with the particular $\mathcal{L}_{NT}$ model of $\mathfrak{N} = (\mathbb{N}, 0, S, +, \cdot, E, <)$ with the usual definitions of these functions.

2. **Completeness with respect to set of sentences** Let Σ be a set of sentences. Then Σ is complete if *every* sentence τ of $\mathcal{L}$ is either provable or it's negation is provable, that is:

$$\Sigma \vdash \tau \qquad \Sigma \vdash \neg\tau$$

The set Σ is complete if and only if $\mathrm{Th}(\Sigma)$ is maximally consistent.

3. **Gödel Codes and Sequences** Each symbol of $\mathcal{L}_{NT}$ is given a unique number, see definition 4.1.2. Then any sequence of symbols of $\mathcal{L}_{NT}$ (including terms and formulas as well as arbitrary sequences) is uniquely inceded as :

$$[s_1, s_2, ..., s_k] = p_1^{[s_1]} \cdots p_k^{[s_k]}$$

4. **Verifying Gödel Codes**: We have the primitive recursive functions *Term*, *Formula*, *Sentence*, and *Logical* to check whether a Gödel code (or in particular any natural number) represents a term, formula, sentence, or logical axiom.

5. **Recursive Set** If Σ is a set of formulas of $\mathcal{L}_{NT}$, Then:

$$[\Delta] = \{[\delta] \mid \delta \in \Delta\}$$

Then Δ is called recursive if $[\Delta]$ is a reursive relation, and Δ is recursively enumerable if $[\Delta]$ is reursively enumerable.

6. **Deductions are Recursive** The following functions are recursive relations that show that the process of deducing is a recursive:

   (a) $\mathit{Premise}_\Delta(n)$: detects whether $n = [\delta]$ is either a logical axiom or in Δ

   (b) $\mathit{Formula}(n)$: detects if $n = [\varphi_1\varphi_2...\varphi_k]$ is a sequence of formulas of $\mathcal{L}_{NT}$

   (c) $\mathit{Inference}(n, i, j)$: Given $n = [\varphi_1\varphi_2...\varphi_k]$, detects when φ_k follows from φ_i and φ_j using Modus ponens and $1 \leq i, j \leq k$

   (d) $\mathit{Deduction}_\Delta(n)$: Given $n = [\varphi_1\varphi_2...\varphi_k]$ detects when this is a valid deduction

   (e) $\mathit{Conclusion}_\Delta(n, m)$ Given $n = [\varphi_1\varphi_2...\varphi_k]$ and $m = [\varphi_k]$, detects when $\Delta \vdash \varphi_k$

7. **Detecting Completeness** If Δ is recursive, then:

   (a) $[\mathrm{Th}(\Delta)]$ is recursively enumerable

   (b) $[\mathrm{Th}(\Delta)]$ is recursive if and only if Δ is complete

8. **Non-logical axioms of NT**: set of axioms we shall use to represent recursive functions, see definition 4.2.2. If this set is labeled $\mathcal{A}$, then $\mathfrak{N} \models \mathcal{A}$ Furthermore, for any assignment function s, $\mathbf{s}(S^m 0) = m$

9. **Representable Function and Relation** Given Σ a set of formulas and a k-place function f, Then f is *representable* in $\mathrm{Th}(\Sigma)$ if

$$f(n_1, n_2, ..., n_k) = m \quad \Leftrightarrow \quad \Sigma \vdash \varphi(S^{n_1}0, ..., S^{n_k}0, S^m 0)$$

In other words, the output of f given $n_1, n_2, ..., n_k$ can be deduced from a formula φ with variables $v_1, v_2, ..., v_k$ replaced with the input of f. This is slightly different for a relation since a relation has no output, we'd instead have:

$$P(n_1, n_2, ..., n_k) = m \quad \Leftrightarrow \quad \Sigma \vdash \varphi(S^{n_1}0, ..., S^{n_k}0)$$

Naturally, doing the usual trick of going back and forth between functions works work representable functions too.

10. **Recursive, then Representable**: First, in example 4.2, we showed that the initial functions are representable. We then showed that composition and unbounded mineralization is representable. We finish off by showing a few key relation and functions are representable to show primitive recursion is representable, making all recursive functions representable

11. **Representability and Consistency**:

    (a) If a function f is representable in $\text{Th}(\Sigma)$, then Σ is consistent(!)

    (b) if Σ, Γ consistent and $\Sigma \vdash \Gamma$, then every function and relation representable in $\text{Th}(\Gamma)$ is representable in $\text{Th}(\Sigma)$

    (c) If Σ is consistent and $\Sigma \vdash \mathcal{A}$, then representable function and relations in $\text{Th}(\mathcal{A})$ are representable in $\text{Th}(\Sigma)$.

12. **Incompleteness Lemma**: $Sub(n, k)$ takes in $n = [\varphi]$ where φ has at most v_1 free and outputs $[\varphi(S^k 0)]$ is recursive, and hence representable in $\text{Th}(\mathcal{A})$. Next, $\mathcal{A}$ is recursive in $\mathcal{L}_{NT}$. Finally, if φ has only one free variable, then there is a sentence such that

$$\mathcal{A} \vdash \sigma \leftrightarrow \varphi(S^{[\sigma]}0)$$

13. **Gödels Incompleteness Theorem I**: Let Σ be a consistent recursive set of sentences of $\mathcal{L}_{NT}$ such that $\Sigma \vdash \mathcal{A}$. Then Σ is *not* complete

14. **Consequence of Thm I**:

    (a) Γ is complete and $\Gamma \cup \mathcal{A}$ is consistent, then Γ is *not* recursive

    (b) Δ recursive and $\Delta \cup \mathcal{A}$ consistent, then Δ *not* complete

    (c) the set $\text{Th}(\mathfrak{N})$ consisting of all sentence where $\mathfrak{N} \models \sigma$ is *not* recursive

15. **Consistency**: Σ recrusvie, then there is a set of sentences $\text{Con}(\Sigma)$ such that Σ is consistent if and only if $\mathcal{A} \vdash \text{Con}(\Sigma)$

16. **Gödels Incompleteness Theorem II**: Σ is consistent and recursive such that $\Sigma \vdash \mathcal{A}$, then $\Sigma \nvdash \text{Con}(\Sigma)$

17. **Definable**: Let P be a k-place relation. Then P is definable if

$$P(n_1, n_2, ..., n_k) \Leftrightarrow \mathfrak{N} \models \varphi[s(v_1|n_1)...(v_k|n_k)]$$

    notice we have an implications $\models$ instead of $\vdash$ and overrides instead of substitutions.

18. **Represent ability and Definability**: If P is representable in $\text{Th}(\mathcal{A})$, then P is definable in $\mathfrak{N}$.

19. **Tarski's Undefinable Theorem**: The set $[\text{Th}(\mathfrak{N})]$ is *not* definable in $\mathfrak{N}$