Any statement that is not an Axiom \((\text{Ax})\) or a Definition \((\text{Def})\) or a Remark \((\text{R})\) or an Example is an Exercise. Unless it says otherwise, the exercise is to show carefully whether the statement is true or false. You’re always allowed to take any earlier results for granted when you are working on later ones. (But not the other way around!)

# 1 Functions and Sets

## 1.1 Sets and Subsets

Even though I’m marking as “Ax” anything that we won’t prove, don’t take this seriously as a set of axioms. These would be kind of a ridiculous set of axioms, because they are very redundant. In a serious set theory course, we’d work through how to derive lots of these facts from each other. But this isn’t a set theory course, so let’s just get this stuff out there so we can take it as common ground. (They’re also not quite as strong as standard set theory.)

We’ll start with some basics about sets—collections of elements. The notation \(a \in A\) means that \(a\) is an element of \(A\). Saying \(A = \{a_1, \ldots, a_n\}\) means that \(a_1, \ldots, a_n\) are all of the elements of \(A\). (Note it doesn’t follow that \(A\) has \(n\) elements! It could be that there’s some \(i\) and \(j\), with \(i \neq j\), such that \(a_i = a_j\). In general, it’s good to remember that just because we’re using two different names, it doesn’t follow that they are names for two different things.)

1. **Def.** \(A\) is a **subset** of \(B\) iff every element of \(A\) is an element of \(B\). For short: \(A \subseteq B\).

2. **R.** “Iff” means “if and only if”. That is, “blah iff zoom” means the same thing as “if blah, then zoom, and if zoom, then blah”.

3. For any set \(A\), \(A\) is a subset of \(A\).

4. For any sets \(A\), \(B\), and \(C\), if \(A\) is a subset of \(B\), and \(B\) is a subset of \(C\), then \(A\) is a subset of \(C\).

5. **Ax** (Extensionality). If \(A\) is a subset of \(B\) and \(B\) is a subset of \(A\), then \(A = B\).
6. **R.** The equals sign means “… is the very same thing as …”.

7. \( A = B \) iff \( A \) and \( B \) have exactly the same elements.

8. If \( A \subseteq B \subseteq C \subseteq A \) then \( A = B = C \).

9. **Ax** (Separation). For any set \( A \), there is a set whose elements are just those elements \( a \) of \( A \) such that \( F(a) \). This set is labeled \( \{ a \in A \mid F(a) \} \).

10. **R.** This is a **schematic** principle, and \( F \) is a schematic variable. You get an instance of it by substituting for the formula \( R(a) \) some precise description with one variable \( a \). In context here it’s really not totally clear what the legitimate instances are, and in fact there are paradoxes that arise from choosing them unwisely. You can alternatively think of it as a **second-order** principle, which quantifies over different properties—or perhaps more precisely, “generalizes in predicate position”. This interpretation is better in some ways—hopefully we’ll return to this by the end of the course. But many people have philosophical objections to second-order quantification.

11. **Example.** If we have a set \( A = \{1, 2, 3, 4, 5\} \), then

\[
\begin{align*}
\{ a \in A \mid a \text{ is prime } \} & = \{2, 3, 5\} \\
\{ a \in A \mid a \text{ is even } \} & = \{2, 4\} \\
\{ a \in A \mid a \geq 5 \} & = \{5\}
\end{align*}
\]  

(1)

12. Suppose that \( a_1, \ldots, a_n \) are elements of \( A \) (though \( A \) may have other elements as well). Then there is a set \( \{ a_1, \ldots, a_n \} \).

13. Show that for any \( A, B \subseteq U \) the following sets exist:

(a) The **union** of \( A \) and \( B \), which contains whatever is in \( A \) or in \( B \) (or both). We label this set \( A \cup B \).

(b) The **intersection** of \( A \) and \( B \), which contains just what \( A \) and \( B \) both contain. We label this set \( A \cap B \).

(c) The **difference** of \( A \) and \( B \), which contains things that are in \( A \) but not \( B \). We label this set \( A - B \).

14. Show that \( A - (B \cup C) = (A - B) \cap (A - C) \).

15. **Ax.** For any set \( A \) there is a **power set** \( PA \) whose elements are all subsets of \( A \).
16. If \( A \) has six elements, how many elements does \( P(A) \) have?

17. **Ax.** There is a set \( \emptyset = \{ \} \) with zero elements, a set \( 1 = \{ * \} \) with exactly one element, and a set \( 2 = \{ \bot, \top \} \) with exactly two elements.

18. Axiom 1.1.17 is redundant. Show how the existence of a set with zero elements and a set with one element follows from the existence of 2.

### 1.2 Functions

1. **Ax.** For any sets \( A \) and \( B \), there is a set \( A \rightarrow B \) of functions with **domain** \( A \) and **codomain** \( B \). This is sometimes also labelled \( B^A \) (mainly when using an arrow would be confusing, because the arrow has multiple meanings). We write \( f : A \rightarrow B \) to say that \( f \) is a function from \( A \) to \( B \). For each function \( f : A \rightarrow B \) and each element \( a \in A \), there is a unique element of \( B \) which is the result of applying \( f \) to \( a \). This is labeled \( fa \).

2. **R.** So these all mean the same thing:
   
   - \( f \in B^A \)
   - \( f : A \rightarrow B \)
   - \( f \in A \rightarrow B \)

   If we write \( A \rightarrow B \rightarrow C \), then the implicit parentheses go to the right: \( A \rightarrow (B \rightarrow C) \), not \( (A \rightarrow B) \rightarrow C \). (We’ll see later why this is convenient.)

   Some people write function application with lots of extra parentheses: \( f(a) \) rather than \( fa \). But I won’t do that unless things would be unclear otherwise.

3. **Ax** (Choice). If for every \( a \in A \) there is some \( b \in B \), such that \( F(a, b) \), then there is some function \( f : A \rightarrow B \) such that for every \( a \in A \), \( P(a, fa) \).

4. **R.** This is another schematic principle.

5. **Ax** (Function Extensionality). For any \( f, g : A \rightarrow B \), if \( fa = ga \) for every \( a \in A \), then \( f = g \).

6. **R.** Choice guarantees that there are enough functions. Extensionality guarantees that there are not too many functions. But Choice is a controversial axiom, for a couple of reasons. First, it has surprisingly strong consequences when it comes to infinite sets. Second, unlike the other axioms it
is non-constructive. We can get functions from Choice that we have no way of describing uniquely.

Choice and Extensionality are often used together in the following way: if for each \( a \in A \) we can pick out a some unique \( b \in B \) that satisfies some description \( F(a, b) \), then there is a unique function \( f : A \to B \) such that \( F(a, fa) \) for every \( a \in A \). (Here’s why \( f \) is unique. Suppose there are functions \( f \) and \( f' \) such that \( F(a, fa) \) and \( F(a, f'a) \). If there is exactly one \( b \in B \) such that \( F(a, b) \), it follows that \( fa = b = f'a \). Since this is true for every \( a \in A \), by Function Extensionality \( f = f' \).)

7. **Example.** For any set \( A \) there is a unique function \( \text{id}_A : A \to A \) such that \( \text{id}_A a = a \) for every \( a \in A \). This is called the **identity function**.

   **Proof.** For every \( a \in A \) there is exactly one \( a' \in A \) such that \( a = a' \) (namely \( a \)). So there is a unique function \( f : A \to A \) such that \( a = fa \) for each \( a \in A \).

8. **Example.** For each \( f : A \to B \) and \( g : B \to C \), there is a unique function \( g \circ f : A \to C \) such that for each \( a \in A \), \( (g \circ f)a = g(fa) \). This is called the **composite** of \( g \) and \( f \).

   **Proof.** It’s clear that for each \( a \in A \) there is exactly one \( c \in C \) such that \( c = g(fa) \).

9. Let \( f : A \to B \). Then \( f \circ \text{id}_A = f \) and \( \text{id}_B \circ f = f \).

10. For any \( f : A \to B, g : B \to C \), and \( h : C \to D \),

\[
(h \circ g) \circ f = h \circ (g \circ f).
\]

1.3 Pairs and Sums

1. **Ax (Products).** For any sets \( A \) and \( B \) there is a set \( A \times B \) of (ordered) **pairs**.

   There is a **first element** function \( \pi_1 : A \times B \to A \), and a **second element** function \( \pi_2 : A \times B \to B \), and for any \( a \in A \) and \( b \in B \) there is a unique element \( (a, b) \in A \times B \) such that the first element of \( (a, b) \) is \( a \) and the second element of \( (a, b) \) is \( b \).

2. How many elements do the following sets have? (Justify your answers.)

   (a) \( 1 \times 2 \)

   (b) \( 2 \times 0 \)
1.4 One-to-One Correspondences

3. **Ax (Sums).** For any sets $A$ and $B$ there is a **sum** (or **disjoint union**) $A + B$. There is a **first inclusion** function $i_1$ from $A$ to $A + B$ and a **second inclusion** function $i_2$ from $B$ to $A + B$, and every element of $A + B$ is either equal to $i_1 a$ for some $a \in A$ or else $i_2 b$ for some $b \in B$, but not both.

4. **R.** Sums aren’t quite as standardly used as products but they can be really useful. It’s possible to construct pairs and sums, or surrogates for them, from other sorts of sets (and this is how it’s done in standard set theory). But that’s none of our business right now.

Remind me to draw some pictures of products and sums for you in case I forget.

5. Axiom 1.1.17 is redundant. Show how the existence of a set with two elements follows from the existence of 1.

6. How many elements do the following sets have? (Justify your answers.)

   (a) $1 + 2$
   (b) $1 \times (0 + 2)$
   (c) $(2 \times (2 + 1)) + 1$

7. There is a set with exactly 256 elements.

1.4 One-to-One Correspondences

1. **Def.** A function $f : A \to B$ is **one-to-one** (or **injective**) iff for each $a, a' \in A$, if $fa = fa'$ then $a = a'$.

2. **Def.** A function $f : A \to B$ is **onto** (or **surjective**) iff for each $b \in B$ there is some $a \in A$ such that $fa = b$.

3. Give an example of a function which is …

   (a) One-to-one but not onto.
   (b) Onto but not one-to-one.
   (c) One-to-one and onto.

4. If $f : A \to B$ and $g : B \to C$ are both one-to-one, then $g \circ f : A \to C$ is one-to-one.
5. If \( f : A \to B \) and \( g : B \to C \) are both onto, then \( g \circ f : A \to C \) is onto.

6. If \( f : A \to B \) is onto, then there is a one-to-one function from \( B \) to \( A \).

7. If \( f : A \to B \) is one-to-one, then there is a function from \( B \) onto \( A \) or else \( A \) is empty.

8. **Def.** A function \( f : A \to B \) is a **one-to-one correspondence** (or a **bijection**) iff it is one-to-one and onto. Write \( A \cong B \) (\( A \) and \( B \) are “similar” or “equinumerous”) iff there is a one-to-one correspondence between \( A \) and \( B \).

9. **Def.** For any function \( f : A \to B \), an **inverse** of \( f \) is a function \( g : B \to A \) such that

\[
\begin{align*}
g \circ f &= \text{id}_A \\
f \circ g &= \text{id}_B
\end{align*}
\]

10. For any \( f : A \to B \), if \( f \) has an inverse then \( f \) is a one-to-one correspondence.

11. For any \( f : A \to B \), if \( f \) is a one-to-one correspondence then \( f \) has an inverse.

12. For any sets \( A, B, C \):

   (a) \( A \cong A \).

   (b) If \( A \cong B \) and \( B \cong C \) then \( A \cong C \).

   (c) If \( A \cong B \) then \( B \cong A \).

13. For any sets \( A \) and \( B \), there is a one-to-one function \( f : A \to B \) iff there is a subset \( S \subseteq B \) such that \( A \cong S \).

14. For any sets \( A, B, C \),

   (a) \( A \times B \cong B \times A \)

   (b) \( A + B \cong B + A \)

   (c) \( 1 \times A \cong A \)

   (d) \( 0 \to A \cong 1 \)

   (e) \( C \to (A \times B) \cong (C \to A) \times (C \to B) \)

   (f) \( (A + B) \to C \cong (A \to C) \times (B \to C) \)

   (g) \( (A \times B) \to C \cong A \to (B \to C) \)

   (h) \( PA \cong A \to 2 \).
15. **R.** It’s often useful to think of functions $A \rightarrow 2$ as properties of elements of $A$, and I’ll sometimes talk about them this way. Similarly, we can think of functions $A \times B \rightarrow 2$ as relations between elements of $A$ and elements of $B$. This idea is the basis for truth tables in propositional logic, as we’ll see later on. Of course, we often think of properties and relations as finer-grained than this—there could be two different properties that apply to exactly the same individuals. (The classic example: “animal with a heart” and “animal with a kidney”. ) But since most of what we’re doing is about abstract objects, and nearly every truth we’ll consider is a necessary and a priori truth, it’s mostly harmless to think of properties and relations in this coarse-grained way.

16. Let $f : A \rightarrow PA$, and let $R = \{ a \in A \mid a \notin fa \}$. Then $R$ is not in the range of $f$.

17. (Cantor’s Theorem.) There is no set $A$ for which $A \cong PA$.

18. There is no set of all sets.

## 2 The Uncountable

### 2.1 Numbers

1. **R.** The natural numbers are the counting numbers 0, 1, 2, …. We’ll start by trying to give a more precise definition. One reason for doing this is that precise definitions will help us to give more precise proofs using numbers. It will also be useful as a model for definitions of other structures later on.

   **Attempted Definition.** The natural numbers are a set $\mathbb{N}$ with an element 0 (the zero element) and a function $\text{suc} : \mathbb{N} \rightarrow \mathbb{N}$ (the successor function, intuitively adding one) such that every element of $\mathbb{N}$ is the result of applying $\text{suc}$ to 0 finitely many times.

   The obvious trouble with this is that it appeals to the notion of “finitely many times”. It seems difficult to make that idea precise without appealing to the idea of a natural number, which would be circular. But it can be done! We’ll use the version codified by Peano and Dedekind. It has two main parts. The first part guarantees that zero and the successors of any distinct numbers are all distinct from each other. The second part uses the idea of proof by induction—which is a tool we will use extensively. The idea of a proof by induction is that we can prove that all natural numbers have a certain property by first proving that 0 has it, and then proving that the property is “inherited” by successors.
2. **Def.** The **natural numbers** are a set \( \mathbb{N} \) with an element \( 0 \in \mathbb{N} \) and a function \( \text{suc} : \mathbb{N} \to \mathbb{N} \) with the following three properties:

(a) 0 is not a successor

(b) For any \( n, n' \in \mathbb{N} \), if \( \text{suc} n = \text{suc} n' \), then \( n = n' \)

(c) Let \( S \subseteq \mathbb{N} \), suppose \( 0 \in S \), and suppose that for each \( n \in \mathbb{N} \), if \( n \in S \) then \( \text{suc} n \in S \). Then \( S = \mathbb{N} \).

3. **R.** These conditions are a version of the **Peano axioms** for arithmetic.

4. **R.** It can be kind of nice to think of these two structures of the natural numbers, \( 0 \in \mathbb{N} \) and \( \text{suc} : \mathbb{N} \to \mathbb{N} \), in terms of one combined structure, which is the function \( s : \mathbb{1} + \mathbb{N} \to \mathbb{N} \) that takes the single object on the left to 0, and the numbers on the right to their successors:

\[
\begin{align*}
    s(i_1 \bullet) &= 0 \\
    s(i_2 n) &= \text{suc} n
\end{align*}
\]

Then conditions (a) and (b) in the definition together say that \( s \) is a one-to-one function.

5. **R.** This definition is still a bit problematic, because we say “the natural numbers”, even though we haven’t shown that there is any set \( \mathbb{N} \) with 0 and \( \text{suc} \) like this. 

We also haven’t shown that it’s **unique.** In fact, it isn’t unique, but rather only “unique up to unique isomorphism”. This means that there are alternative structures that differ merely in what particular objects the elements 0, \( \text{suc} 0 \), \( \text{suc} \text{suc} 0 \), … are. Structures like this, which are isomorphic to the natural numbers, are called **\( \omega \)-sequences.** But let’s not get sidetracked with this issue right now.

In fact, we **can’t** prove that there is a set of natural numbers using the assumptions we already have—those axioms are consistent with every set being finite. So we’ll just take it as another assumption—the Axiom of Infinity.

6. **Ax (Infinity).** There is a set \( \mathbb{N} \) of natural numbers that satisfies Def. 2.1.2.

7. For every \( n \in \mathbb{N} \), either \( n = 0 \) or \( n = \text{suc} m \) for some \( m \in \mathbb{N} \).

8. There is no \( n \in \mathbb{N} \) for which \( n = \text{suc} n \).
9. **R.** Another useful tool, which is closely related to induction, is **recursive definition** of functions. Say you want to specify an **infinite sequence** in a set $A$—which we can think of as a function $f : \mathbb{N} \to A$. One way to do this is to give the first element $f(0) \in A$—the “initial condition”—and then give a rule for how to produce $f(n)$ from $f(n)$—the “law of evolution”. It’s a fact that there is always a unique sequence for any given initial condition and law of evolution. We won’t take the time to prove this fact right now (or even precisely state its general version), but let’s consider some examples. (We’ll come back to this issue in more detail in Part 6.)

10. **Example.** There is a unique function $f : \mathbb{N} \to \mathbb{N}$ (**doubling**) defined such that

$$
    f(0) = 0
$$

$$
    f(suc\ n) = suc\ suc\ f(n)
$$

This is the sequence $0, 2, 4, 6, \ldots$.

11. **Example.** There is a unique function $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (**addition**) defined such that (for every $m, n \in \mathbb{N}$):

$$
    0 + n = n
$$

$$
    suc\ m + n = suc(m + n)
$$

12. **Example.** There is a unique function $\cdot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (**multiplication**) defined such that (for every $m, n \in \mathbb{N}$):

$$
    0 \cdot n = 0
$$

$$
    suc\ m \cdot n = m \cdot n + n
$$

13. **Example.** There is a unique function $\uparrow : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (**exponentiation**) defined such that (for every $m, n \in \mathbb{N}$):

$$
    n \uparrow 0 = 1
$$

$$
    n \uparrow suc\ m = n \cdot (n \uparrow m)
$$

14. **Example.** There is a unique function factorial : $\mathbb{N} \to \mathbb{N}$ defined such that (for every $n \in \mathbb{N}$)

$$
    \text{factorial}(0) = 1 = suc\ 0
$$

$$
    \text{factorial}(suc\ n) = n \cdot \text{factorial}\ n
$$

This gives the sequence $1, 1, 2, 6, 24, \ldots$. 
15. Let the number 1 be defined to be $\text{suc} 0$. Then $\text{suc} n = 1 + n$.

16. Show the following by induction, for every $k, m, n \in \mathbb{N}$.

   (a) **Example.** $n + 0 = n$.
       **Proof.** By induction. We need to show two things: the “base case”
       (zero has the property), and the “inductive step” (successors inherit the
       property).
       i. **Base case.** $0 + 0 = 0$
          This follows immediately from the definition of $+$.
       ii. **Inductive step.** For any $n \in \mathbb{N}$, if $n + 0 = n$, then $\text{suc} n + 0 = \text{suc} n$.
          By the definition of addition, $\text{suc} n + 0 = \text{suc}(n+0)$. By hypothesis,
          $\text{suc}(n+0)$ is the same as $\text{suc} n$.

   (b) **Example.** $n + 1 = \text{suc} n$.
       **Proof.** By induction. There are two steps.
       i. **Base case.** $0 + 1 = \text{suc} 0$.
          By the definition of addition, $0 + 1 = 1$, which is $\text{suc} 0$ by definition.
       ii. **Inductive step.** If $n + 1 = \text{suc} n$, then $\text{suc} n + 1 = \text{suc}(\text{suc} n)$.
          $\text{suc} n + 1 = \text{suc}(n+1)$ (by the definition of addition), and by hy-
          pothesis this is $\text{suc} \text{suc} n$.

   (c) Addition is **associative**: $(k + m) + n = k + (m + n)$.
   (d) Addition is **commutative**: $m + n = n + m$.
   (e) Addition is **cancellative**: if $k + m = k + n$ then $m = n$.

17. If $m + n = 0$ then $m = n = 0$.

18. **Def.**

   (a) $m \leq n$ if there is some $k \in \mathbb{N}$ such that $k + m = n$.
   (b) $m \geq n$ if $n \leq m$.
   (c) $m < n$ if $m \leq n$ and $m \neq n$.

19. Show the following, for every $k, m, n \in \mathbb{N}$.

   (a) $\leq$ is **reflexive**: $n \leq n$.
   (b) $\leq$ is **transitive**: if $k \leq m$ and $m \leq n$ then $k \leq n$.
   (c) $\leq$ is **antisymmetric**: if $m \leq n$ and $n \leq m$ then $m = n$. 
2.2 Finiteness and Infinity

1. **Def.** For each $n \in \mathbb{N}$ let $n = \{k \in \mathbb{N} \mid k < n\}$. (Sometimes it’s easier to just drop the double-struck font and call the set $n$, when that’s not too confusing.)

2. If $k = m + n$, then $k \cong m + n$. In other words, there is a one-to-one correspondence between these sets:

$$\{a \in \mathbb{N} \mid a < m + n\} \quad (10)$$

$$\{a \in \mathbb{N} \mid a < m\} + \{a \in \mathbb{N} \mid a < n\} \quad (11)$$

3. **Def.** A set $A$ has **at most $n$ elements** (for $n \in \mathbb{N}$) iff there is some one-to-one function $f : A \to n$. $A$ is **finite** iff there is some $n$ such that $A$ has at most $n$ elements. Otherwise $A$ is **infinite**.

(a) $A$ has at most $n$ elements iff there is a function from $n$ onto $A$.

**Hint.** Remember 2.4.6 and 2.4.7.
4. If \( f : A \to B \) is one-to-one and \( B \) has at most \( n \) elements, then \( A \) has at most \( n \) elements. (Therefore, if \( B \) is finite, then \( A \) is finite; and if \( A \) is infinite, then \( B \) is infinite.)

5. If \( f : A \to B \) is onto and \( A \) has at most \( n \) elements, then \( B \) has at most \( n \) elements. (Therefore, if \( A \) is finite, then \( B \) is finite; and if \( B \) is infinite, then \( A \) is infinite.)

6. This one has several parts.
   
   (a) For any one-to-one function \( f : A + 1 \to B + 1 \) there is some one-to-one function from \( A \) to \( B \).
   
   \( \text{Hint.} \) There are two cases to consider, depending on whether \( i_2 \bullet \) is in the range of \( f \) or not.
   
   (b) Suppose that \( k > 0 \). There is no one-to-one function from \( k + n \) to \( n \).
   
   \( \text{Hint.} \) Use induction on \( n \) (or equivalently the Least Number Principle).
   
   (c) If there is a one-to-one function from \( m \) to \( n \), then \( m \leq n \).
   
   (d) If \( m \cong n \) then \( m = n \).

7. There is no \( n \in \mathbb{N} \) such that \( 1 + n \cong n \).

8. So does this one.
   
   (a) Suppose \( f : A \to B + 1 \) is one-to-one but not onto. Then there is a one-to-one function from \( A \) to \( B \).
   
   (b) If \( A \) is finite, then there is some \( n \) such that \( A \cong n \).
   
   \( \text{Hint.} \) Since \( A \) is finite, the set of numbers \( m \) for which one-to-one functions from \( A \) to \( m \) exist is non-empty. Use the Least Number Principle.

9. \textbf{Def.} By 7 and 8, for each finite set \( A \) there is exactly one number \( n \in \mathbb{N} \) such that \( A \cong n \). This is the number of elements of \( A \), also called the cardinal number of \( A \).

10. There is no finite set \( A \) such that \( 1 + A \cong A \).

11. (Hilbert’s Hotel.) \( 1 + \mathbb{N} \cong \mathbb{N} \).

12. \( \mathbb{N} \) is infinite.
2.3 Countable Infinity

1. **Definition.** $A$ is **countable** (or **enumerable**) iff there is a one-to-one function $A \rightarrow \mathbb{N}$. Otherwise it is **uncountable**. (A **countably infinite** set is one that is both countable and infinite.)

2. Every finite set is countable.

3. The set of **even** numbers $E$ is countably infinite, where

   $$E = \{ n \in \mathbb{N} | n = m + m \text{ for some } m \in \mathbb{N} \} \quad (12)$$

4. $\mathbb{N} + \mathbb{N}$ is countable.

5. If $B$ is countable and $f : A \rightarrow B$ is one-to-one then $A$ is countable.

6. For any one-to-one functions $A \rightarrow C$ and $B \rightarrow D$, there is a one-to-one function $A + B \rightarrow C + D$.

7. If $A$ and $B$ are countable, then $A + B$ is countable.

8. If $A \cong \mathbb{N}$ then $A$ is countably infinite.

9. **R.** The converse is also true, but showing that takes a bit more work. Suppose we can assign numbers $fa$ to the elements $a \in A$. This assignment might leave gaps, numbers with no $a \in A$ assigned to them. We can make the assignment into a one-to-one correspondence by closing up the gaps—shifting each element of $A$ to the furthest unoccupied spot to the left. In that case, the index we assign to each $a \in A$ is the number of elements of $A$ with a lower $f$-number than $a$.

10. Suppose $A$ is infinite and $f : A \rightarrow \mathbb{N}$ is one-to-one. For each $a \in A$, let $La = \{ a' \in A | fa' < fa \}$. This is the **lower set** for $a$.

    (a) For each $a \in A$, $La$ is finite. Let $ga$ be the number of elements of $La$.

    (b) There is exactly one $a_0 \in A$ such that $ga_0 = 0$.

    **Hint.** The range of $f$ has a least element.

    (c) For each $a \in A$ there is exactly one $a' \in A$ such that $ga' = \text{suc } ga$.

    **Hint.** For each $a \in A$, this set has a least element:

    $$\{ n \in \mathbb{N} | \text{ there is some } a' \in A \text{ such that } fa' = n \text{ and } n > fa \} \quad (13)$$
(d) If $A$ is any countably infinite set, then $A \cong \mathbb{N}$.

11. **R.** So we’ve shown both directions of this fact: $A$ is countably infinite iff $A \cong \mathbb{N}$. This is often a useful way to think about what countably infinite sets are: they are sets that “look like” the set of natural numbers, as far as facts about the existence of one-to-one functions go.

### 2.4 Finite Sequences

1. **Def.** For $n \in \mathbb{N}$, a length-$n$ **sequence** in $A$ is a function $n \rightarrow A$. Thus $A^n$ (recall that this is an alternative notation for the set of functions $n \rightarrow A$) is the set of all length-$n$ sequences. We usually just write this as $A^n$. We can label a finite sequence in $A^n$ by listing its values $(a_0, \ldots, a_{n-1})$.

2. **Ax.** There is a set $A^*$ of all **finite sequences** in $A$. That is, $x \in A^*$ iff there is some $n \in \mathbb{N}$ such that $x \in A^n$.

3. **R.** Like some of our other axioms, this one is sort of silly: we could pretty easily construct a set of surrogates for finite sequences from the materials we’ve already assumed. (For instance, we could use infinite sequences in $A + 1$ and use the extra symbol $i_2\bullet$ to mark the end of the sequence.) Time’s winged chariot, etc.

4. For any set $A$,
   
   (a) $A^0 \cong 1$
   
   (b) $A^1 \cong A$
   
   (c) $A^2 \cong A \times A$
   
   (d) $A^{n+1} \cong A \times A^n$ for each $n \in \mathbb{N}$.

   **Hint.** You can do these by directly constructing one-to-one correspondences. Alternatively, there are slick proofs that use facts from 1.4.14.

5. $1^* \cong \mathbb{N}$.

6. **R.** The correspondence in exercise 5 is sometimes called “unary” (one-digit) or “stroke” or “tally” notation.

7. (Binary Notation.)
2.4 Finite Sequences

(a) Let \( 2 = \text{suc suc } 0 \). First show that for any \( m \in \mathbb{N} \), \( m + m = 2 \cdot m \).

Conclude from this that for any \( n \in \mathbb{N} \),

\[
(2 \uparrow n) + (2 \uparrow n) = 2 \uparrow (n + 1). \tag{14}
\]

(b) For each \( n \in \mathbb{N} \) there is a one-to-one function \( f_n : 2^{n+1} \rightarrow \mathbb{N} \) such that, for any sequence \((b_0, \ldots, b_n)\), if \( b_n = 0 \) then

\[
f_n(b_0, \ldots, b_n) < 2 \uparrow n \tag{15}
\]

and if \( b_n = 1 \) then

\[
2 \uparrow n \leq f_n(b_0, \ldots, b_n) < 2 \uparrow (n + 1) \tag{16}
\]

*Hint.* Use induction on \( n \).

(c) The set of finite sequences in \( 2 \) which end in one is countable.

(d) \( 2^* \) is countable.

8. There is a one-to-one function \( A^* \times B^* \rightarrow (A + B)^* \).

9. \( \mathbb{N} \times \mathbb{N} \) is countable.
   *Hint.* Use 5, 7 and 8.

10. **R.** Can you describe explicitly what the one-to-one function \( \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \) is that you used in your proof?

   These exercises walked us through one way of proving that the set of pairs of numbers is countable, but there are lots of other creative coding systems for doing it. See BBJ Ch. 1 for a couple of alternative proofs.

11. If \( A \) and \( B \) are countable, then \( A \times B \) is countable.

12. If \( A \) is countable, then \( A^n \) is countable for each \( n \in \mathbb{N} \).
   *Hint.* Use 11 and induction.

13. Suppose that there is a **countable partition** of a set \( A \): there are some subsets \( A_0, A_1, A_2, \ldots \) of \( A \) such that for every \( a \in A \) there is exactly one \( n \in \mathbb{N} \) such that \( a \in A_n \). Suppose also that \( A_n \) is countable for each \( n \in \mathbb{N} \). Then \( A \) is countable.
   *Hint.* Use the fact that for each \( n \in \mathbb{N} \) there is some one-to-one function \( f_n : A_n \rightarrow \mathbb{N} \), together with the fact that the sets \( A_n \) are a partition of \( A \), to define a one-to-one function from \( A \) to \( \mathbb{N} \times \mathbb{N} \).

14. If \( A \) is countable, then \( A^* \) is countable.
2.5 Uncountable Infinity

1. \( \mathbb{P} \) is uncountable.
   
   \textit{Hint.} Remember Cantor’s Theorem.

2. For any set \( A \), if the set \( A^{\mathbb{N}} \) of infinite sequences in \( A \) is countable, then \( A \) has less than two elements.
   
   \textit{Hint.} Fact 1.4.14h will help.

3. Let \( A \) be any countable set—the alphabet. Suppose that there is a set of descriptions \( D \), such that every description is a finite sequence of elements of \( A \). Suppose also that there is a function that takes each description in \( D \) to some \( \mathbb{N} \to \mathbb{N} \) function. (For example, \( A \) could be the set of English letters plus a space, and descriptions could be sequences of letters and spaces that form grammatical English phrases, like “the function that takes each number to its successor.”) Show that there are \( \mathbb{N} \to \mathbb{N} \) functions with no corresponding description.

3 Propositional Logic

3.1 The Language of Propositional Logic: Syntax

1. \( \mathbf{R.} \) Intuitively, the language of propositional logic contains certain basic sentence letters, and also anything you get by repeatedly putting together sentences using logical connectives like “… and …” and “if … then …”, finitely many times. Because it’s convenient for making our proofs simpler, we’ll use a simplified set of logical connectives. (As we’ll see, these are adequate for defining all the other connectives. In fact, we’re really using one more connective than is strictly necessary).

   \textbf{Attempted Definition.} Let \( S \) be a set, which we’ll call the set of sentence letters (or atoms). The propositional language with these sentence letters, which we’ll call \( \text{Prop } S \) for short, has the following properties. (We call elements of \( \text{Prop } S \) sentences.)

   i. Every sentence letter \( s \in S \) is a sentence.
   ii. There is a sentence \( \perp \), the contradiction (or absurdity).
   iii. For any sentences \( A \) and \( B \), the conjunction of \( A \) and \( B \), written \( A \land B \), is a sentence.
iv. For any sentences \( A \) and \( B \), the **conditional** of \( A \) and \( B \), written \( A \rightarrow B \), is a sentence.

v. Every element of \( \text{Prop} \ S \) is the unique result of applying some of these steps finitely many times.

The trouble with this is mainly in the last clause, which isn’t very precise. The idea of repeating certain operations finitely many times suggests that our more official definition should be recursive—like the official definition of the natural numbers.

2. **Def.** For a set \( S \) of **sentence letters**, the **propositional language** \( \text{Prop} \ S \) has the following structure:

   (a) Every sentence letter \( s \in S \) is in \( \text{Prop} \ S \).

   (b) There is an element \( \bot \in \text{Prop} \ S \).

   (c) There is a function from \( \text{Prop} \ S \times \text{Prop} \ S \) to \( \text{Prop} \ S \) that takes \( A \) and \( B \) to the conjunction \( A \land B \).

   (d) There is a function from \( \text{Prop} \ S \times \text{Prop} \ S \) to \( \text{Prop} \ S \) that takes \( A \) and \( B \) to the conditional \( A \rightarrow B \).

   These have the following properties:

   (a) Sentence letters, the contradiction, conjunctions, and conditionals are all distinct from each other. That is, for any sentence letter \( s \), and any sentences \( A, B, A' \) and \( B' \) in \( \text{Prop} \ S \), where \( A \neq A' \) and \( B \neq B' \), all of the following are distinct from one another: \( s, \bot, A \rightarrow B, A' \rightarrow B', A \land B, \) and \( A' \land B' \).

   (b) Consider any subset \( X \subseteq \text{Prop} \ S \). Suppose (i) each sentence letter is in \( X \), (ii) the contradiction \( \bot \) is in \( X \), (iii) for any sentences \( A \in X \) and \( B \in X \), the conjunction \( A \land B \) is in \( X \), and (iv) for any sentences \( A \in X \) and \( B \in X \), the conditional \( A \rightarrow B \) is in \( X \). Given these four facts, it follows that \( X = \text{Prop} \ S \).

   We call the elements of \( \text{Prop} \ S \) **sentences**.

3. **R.** Like with the natural numbers, the final property of sentences tells us that we can prove things about *all* sentences by **structural induction**. To show that every sentence has a certain property \( P \), here’s what we have to do. First, show that the sentence letters have \( P \); second, show that the absurdity has \( P \); third and fourth, show that conjunctions and conditionals inherit \( P \) from
their constituents. (Note that while induction on the natural numbers only requires two cases, for zero and successor, structural induction on sentences requires four cases.)

This also allows us to define a function whose domain is Prop $S$ by \textbf{structural recursion}. To define $f : \text{Prop} \, S \rightarrow X$, all we have to do is say what $f$ should assign to each sentence letter, what $f$ should assign to the contradiction, what $f$ should do to a conjunction $A \land B$ \textit{given} the results $f(A)$ and $f(B)$, and similarly for the conditional. We’ll see this more explicitly in a moment.

4. \textbf{R}. Like the case of the natural numbers, it can be nice to think of the four different kinds of structure on Prop $S$ in terms of one big combined operation: a function

$$O : S + 1 + (\text{Prop} \, S \times \text{Prop} \, S) + (\text{Prop} \, S \times \text{Prop} \, S) \rightarrow \text{Prop} \, S \quad (17)$$

In other words, $O$ takes four kinds of input: in case 1, it takes each sentence letter to itself, in case 2, $O$ takes the single element of 1 to $\bot$, in case 3, $O$ takes a pair of sentences $(A, B)$ to $A \land B$, and in case 4, $O$ takes a pair of sentences $(A, B)$ to $A \rightarrow B$. Then we can state the first property (the \textit{injectivity} property) of all these structures much more concisely: $O$ is a one-to-one function.

5. \textbf{R}. Like the case of the natural numbers, this definition isn’t really proper on its own, because saying “the” implies existence and uniqueness. (And again, Prop $S$ is really only “unique up to unique isomorphism”, rather than uniqueness strictly speaking. Any other set that satisfies the definition basically just relabel the different connectives.) Again, rather than taking the time to prove existence, we’ll just take it as an axiom:

6. \textbf{Ax}. For any set $S$, there is a propositional language Prop $S$.

7. \textbf{R}. Like many of our other existence axioms, it isn’t really necessary to take this as an axiom: we can prove that there is a set like this (at least if $S$ is countable, which is the main case we’ll be worried about). We’ll consider later how to do this, by looking at how to encode sentences using strings of symbols from a finite alphabet. But let’s not get bogged down with this issue right now.

8. Prove the following by structural induction: for any sentence $A$ of Prop $S$, either (i) $A$ is a sentence letter, or (ii) $A$ is $\bot$, or (iii) $A$ is a conjunction $B \land C$ for some $B, C \in \text{Prop} \, S$, or (iv) $A$ is a conditional $B \rightarrow C$ for some $B, C \in \text{Prop} \, S$.

\textit{Hint}. Compare 2.1.7.
9. For any sentence \( A \in \text{Prop} S \), the sentence \( A \land A \) is distinct from \( A \).

10. For any sentences \( A, B, C \in \text{Prop} S \), the sentence \( (A \land B) \land C \) is distinct from \( A \land (B \land C) \).

11. Give a recursive definition of a function which gives the complexity of each sentence \( A \in \text{Prop} S \): intuitively, this is the number of connectives that occur in it. Here are some examples, where \( p, q, r \) are sentence letters:

   - The complexity of a sentence letter \( p \) is zero.
   - The complexity of \( (p \land q) \land p \) is two.
   - The complexity of \( (p \rightarrow \bot) \land p \) is three.

12. (The Least Complexity Principle.) For any non-empty set of sentences \( X \subseteq \text{Prop} S \), there is an element of \( X \) that has minimal complexity: that is, there is some sentence \( A \in X \) such that for every \( B \in X \), the complexity of \( B \) is greater than or equal to the complexity of \( A \).

13. For any set \( S \), any sentence \( A \in \text{Prop} S \) contains only finitely many sentence letters.

   \textit{Hint.} Suppose there exist sentences that contain infinitely many sentence letters, and use the Least Complexity Principle to derive a contradiction.

14. For any set \( S \) and any natural number \( n \), let \( P_n \) be the set of all sentences in \( \text{Prop} S \) with complexity at most \( n \).

   (a) Let \( C_n \) be the set of conjunctions in \( P_n \). There is a one-to-one function from \( C_n \) to \( P_{n-1} \times P_{n-1} \).

   (b) If \( S \) is countable, then \( P_n \) is countable for every \( n \in \mathbb{N} \).

      \textit{Hint.} Suppose \( n \) is the smallest number for which \( P_n \) is uncountable. Show that in that case \( P_n \) can be partitioned into four countable subsets.

   (c) If \( S \) is countable, then \( \text{Prop} S \) is countable.

3.2 Substitution

1. \textbf{R.} It's often useful to think of the sentence letters in \( \text{Prop} S \) as \textbf{propositional variables}. That is, they're placeholders in a sentence which we can “fill in” with more complicated sentences. For example, consider the sentence \( p \rightarrow p \) in \( \text{Prop}\{p\} \). Then consider the sentence \( q \land (q \rightarrow r) \) in \( \text{Prop}\{q, r\} \).
We can substitute this sentence in for \( p \) in the first sentence to make a more complicated sentence:

\[
(q \land (q \rightarrow r)) \rightarrow (q \land (q \rightarrow r))
\]

(18)

Let’s generalize this idea.

Note that it doesn’t really matter which particular objects we choose as sentence letters. So when we’re looking at sentences made up from a finite set of \( n \) sentence letters, it can be convenient to just let the sentence letters be elements of \( n \). Because it looks a little funny to insert numbers into our formulas, I’ll sometimes use \( p_1, \ldots, p_n \) as labels for the elements of \( n \) for this purpose, or sometimes just \( p, q, r \) if \( n \) is a small number.

2. **Def.** For any sentence \( A \in \text{Prop } n \), and any sequence of \( n \) sentences \( B_1, \ldots, B_n \in \text{Prop } S \), there is a substitution instance \( A[B_1, \ldots, B_n] \), which is a sentence in \( \text{Prop } S \), and which is recursively defined as follows:

(a) For each sentence letter, \( p_i[B_1, \ldots, B_n] = B_i \).

(b) \( \bot[B_1, \ldots, B_n] = \bot \).

(c) \( (A_1 \land A_2)[B_1, \ldots, B_n] = A_1[B_1, \ldots, B_n] \land A_2[B_1, \ldots, B_n] \).

(d) \( (A_1 \rightarrow A_2)[B_1, \ldots, B_n] = A_1[B_1, \ldots, B_n] \rightarrow A_2[B_1, \ldots, B_n] \).

3. Show that for any sentence \( A \) with \( n \) sentence letters and any sentences \( B_1, \ldots, B_n \) in \( \text{Prop } S \), the complexity of \( A[B_1, \ldots, B_n] \) is at least the complexity of \( A \).

4. Prove the following facts about substitution.

(a) For any \( A \in \text{Prop } n \),

\[
A[p_1, \ldots, p_n] = A
\]

(19)

(b) Suppose \( A \in \text{Prop } 1 \), \( B \in \text{Prop } 1 \), and \( C \in \text{Prop } S \) for some set \( S \). Then

\[
A[B(C)] = (A[B])[C]
\]

(20)

(It might be helpful to first work through an example to get a better idea of what this means.)

(c) * More generally, substitution is **associative**, in the following sense. Suppose \( A \in \text{Prop } n \), there are \( n \) sentences \( B_1, \ldots, B_n \in \text{Prop } m \), and there are \( m \) sentences \( C_1, \ldots, C_m \in \text{Prop } S \) for some set \( S \). Then:

\[
A[B_1[C_1, \ldots, C_m], \ldots, B_n[C_1, \ldots, C_m]] = A[B_1, \ldots, B_n][C_1, \ldots, C_m]
\]

(21)
3.3 Truth Functions

1. Def.

(a) \(2\) is the set of truth values. We call \(\top\) True and \(\bot\) False.

(b) For any set \(S\), a function from \(S\) to \(2\) (that is, an element of \(2^S\)) is a truth assignment. Sometimes we also call it a model for the propositional language \(\text{Prop} S\).

(c) For any \(n\), a function from \(2^n\) to \(2\) is an \(n\)-place truth function (also called a Boolean operation).

(d) More generally, for any set \(S\), a function from \(2^S\) to \(2\) (that is, a function from truth-assignments to truth-values) is a truth function. We’ll use \(\text{Bool} S\) as a label for the set of all truth functions \(2^S \to 2\).

2. Example. \((\neg) \in \text{Bool} 1\) (negation) is the unique one-place truth function such that

\[
\neg \bot = \top \\
\neg \top = \bot.
\]  

(22)

3. Example. \((\land) \in \text{Bool} 2\) (conjunction) is the unique two-place truth function such that for any truth values \(b_1\) and \(b_2\),

\[
b_1 \land b_2 = \top \text{ iff } b_1 = b_2 = \top.
\]  

(23)

4. Example. \((\lor) \in \text{Bool} 2\) (disjunction) is the unique two-place truth function such that for any truth values \(b_1\) and \(b_2\),

\[
b_1 \lor b_2 = \bot \text{ iff } b_1 = b_2 = \bot.
\]  

(24)

5. Example. \((\rightarrow) \in \text{Bool} 2\) (material implication) is the unique two-place truth function such that for any truth values \(b_1\) and \(b_2\),

\[
b_1 \rightarrow b_2 = \bot \text{ iff } b_1 = \top \text{ and } b_2 = \bot.
\]  

(25)

6. For each of these examples, write out a table to show exactly what value the truth function takes for each possible input.

7. Show that \(\text{Bool} 0 \cong 2\).
8. **Example.** Because $\text{Bool} \ 0 \cong 2$, we also use $\bot$ as a label for the zero-place truth function in $\text{Bool} \ 0$ that corresponds to the truth value $\bot$ in 2, and $\top$ as a label for the zero-place truth function in $\text{Bool} \ 0$ that corresponds to the truth value $\top$ in 2.

9. **Example.** For any set $S$, there is a truth function in $\text{Bool} \ S$ which takes each truth-assignment $M$ to True iff for every $s \in S$, $Ms$ is True.

### 3.4 Interpretations

1. **R.** The propositional language can be interpreted, so that we understand each sentence as a description of a certain truth function. If we think of an assignment of truth values to the sentence letters as a model of a “possible world”, then the interpretation function intuitively says for each sentence what its truth value is in each possible world.

2. **Def.** The **interpretation function** is a function $\langle \rangle : \text{Prop} \ S \rightarrow \text{Bool} \ S$, which is to say that for each sentence $A$ in $\text{Prop} \ S$, and for each truth-assignment $M$ in $2^S$, the interpretation picks out a truth value $\langle A \rangle_M$. This function is defined recursively. For each truth-assignment $M$,

   (a) $\langle s \rangle_M = Ms$ for each sentence letter $s$
   (b) $\langle \bot \rangle_M = \bot$
   (c) $\langle A \land B \rangle_M = \langle A \rangle_M \land \langle B \rangle_M$
   (d) $\langle A \rightarrow B \rangle_M = \langle A \rangle_M \rightarrow \langle B \rangle_M$

(Note that the last three clauses each uses a symbol in two different senses: once as a sentential connective in the language $\text{Prop} \ S$, and once to designate a certain truth function, according to the definitions in the previous section.)

A sentence $A$ denotes the truth function $f$ iff $\langle A \rangle = f$. (That is, $\langle A \rangle_M = f_M$ for every truth assignment $M$.)

3. Write out a table of inputs and outputs for the truth function in $\text{Bool} \ 2$ which is denoted by $p \land (q \rightarrow \bot)$ (where $p$ and $q$ are sentence letters).

4. For any sentence $A$ in $\text{Prop} \ S$, the sentence $A \rightarrow A$ denotes the truth-function which is True for every model. (Extending our notation even more, we also use $\top$ as a label for this constantly-true function in $\text{Bool} \ S$.)

5. For any sentence $A \in \text{Prop} \ S$ and for any truth assignment $M$, $\langle A \rangle_M$ is True iff $\langle A \rightarrow \bot \rangle_M$ is False.
6. Write a sentence in \(\text{Prop} \ n\) (for the appropriate \(n\)) that denotes each of the following functions (defined in the previous section):

(a) **Example.** The material implication function \(\rightarrow\) in Bool 2.

**Solution.** The sentence \(p \rightarrow q\) denotes \(\rightarrow\). To show this, let \(M \in 2^2\). So \(M = (M(p), M(q))\). (Remember that \(p\) and \(q\) are just the first and second elements of 2.) This means that \((\rightarrow)M = M(p) \rightarrow M(q)\). Now apply the definition of the interpretation function:

\[
\llbracket p \rightarrow q \rrbracket_M = \llbracket p \rrbracket_M \rightarrow \llbracket q \rrbracket_M = M(p) \rightarrow M(q) = (\rightarrow)M \]

(b) \(\bot\) in Bool 0

(c) The function in Bool 1 that takes each one-place assignment \((b)\) to the truth-value \(b\).

(d) \(\top\) in Bool 0

(e) The negation function \((\neg)\) in Bool 1

(f) The disjunction function \((\lor)\) in Bool 2

7. Suppose \(f\) is an \(n\)-place truth function, and consider any sequence of \(n\) \(m\)-place truth functions \((g_1, \ldots, g_n) \in \text{Bool} \ m\). Then we define the composite truth function \(f*(g_1, \ldots, g_n)\) to be the \(m\)-place truth function such that for each truth assignment \(M \in 2^m\)

\[
(f*(g_1, \ldots, g_n))M = f(g_1M, \ldots, g_nM) \tag{27}
\]

(a) Suppose \(A\) denotes \(f \in \text{Bool} \ 1\) and \(B\) denotes \(g \in \text{Bool} \ n\). Then the substitution instance \(A[B]\) denotes \(f* (g)\). In brief,

\[
\llbracket A[B] \rrbracket = \llbracket A \rrbracket \ast (\llbracket B \rrbracket) \tag{28}
\]

**Hint.** Use structural induction on \(A\).

(b) More generally, if \(A \in \text{Prop} \ n\) and \(B \in \text{Prop} \ m\), then

\[
\llbracket A[B_1, \ldots, B_n] \rrbracket = \llbracket A \rrbracket \ast (\llbracket B_1 \rrbracket, \ldots, \llbracket B_n \rrbracket) \tag{29}
\]

8. (Substitution of Logical Equivalents.) For any sentences \(A\) and \(B\), say \(A \equiv B\) iff \(A\) and \(B\) denote the same truth-function, i.e. \(\llbracket A \rrbracket = \llbracket B \rrbracket\). Then for any sentences \(A \in \text{Prop} \ 1\), and \(B, C \in \text{Prop} \ S\), if \(B \equiv C\) then \(A[B] \equiv A[C]\).
9. (Expressive Completeness.) For any \( n \)-place truth functions \( g \) and \( h \), let \( g \oplus h \) be the \((n + 1)\)-place truth function such that, for every truth-assignment \((b_1, \ldots, b_{n+1})\) in \(2^{n+1}\),

\[
(g \oplus h)(b_1, \ldots, b_n, b_{n+1}) = \begin{cases} 
  g(b_1, \ldots, b_n) & \text{if } b_{n+1} = \top \\
  h(b_1, \ldots, b_n) & \text{if } b_{n+1} = \bot \end{cases}
\]  

(a) For any \( f \) in \( \text{Bool}(n + 1) \) there are some functions \( g \) and \( h \) in \( \text{Bool} n \) such that \( f = g \oplus h \).

(In fact, this provides an inverse to \( \oplus \), showing that we have one-to-one correspondence:

\[
\text{Bool}(n + 1) \cong \text{Bool} n \times \text{Bool} n.
\]  

It might be illuminating to write this out in terms of the definition of \( \text{Bool} n \) to check it.)

(b) Suppose that \( A \) and \( B \) denote \( n \)-place truth functions \( g \) and \( h \) (respectively). Then there is some sentence which denotes the \((n + 1)\)-place truth-function \( g \oplus h \). (Put more briefly, for any sentences \( A \) and \( B \), there is some sentence \( C \) such that \( \llbracket C \rrbracket = \llbracket A \rrbracket \oplus \llbracket B \rrbracket \).)

(c) For every truth function \( f \in \text{Bool} n \) there is some sentence in \( \text{Prop} n \) that denotes \( f \). (In other words, the interpretation function from \( \text{Prop} n \) to \( \text{Bool} n \) is onto.)

**Hint.** Induction on \( n \)!

10. **R.** We could go on to explore what other languages are functionally complete for Boolean functions; but no time for that! See Sider for some discussion. But note at least the following fact…

11. Give a sentence in \( \text{Prop} 2 \) containing no conjunctions which denotes the truth function \((\land) \in \text{Bool} 2\). Conclude that for every truth function \( f \) in \( \text{Bool} n \) there is a sentence in \( \text{Prop} n \) without any conjunctions that denotes \( f \).

12. * If \( S \) is countably infinite, there is a truth function \( f \) in \( \text{Bool} S \) such that no sentence in \( \text{Prop} S \) denotes \( f \).

**Hint.** You can show this using facts about countable and uncountable sets.
3.5 Validity

1. Def.

(a) A sentence \( A \) is true in a model \( M \) iff \( \langle A \rangle_M \) is True. We also say \( M \) satisfies \( A \), or \( M \) is a model of \( A \), in this case.

(b) Sentences \( A \) and \( B \) are equivalent, written \( A \equiv B \), iff \( A \) and \( B \) denote the same truth-function.

(c) A sentence \( A \) is valid iff \( A \) denotes \( \top \), the truth function which is true in every model.

(d) For a set of sentences \( X \) and a sentence \( B \), \( B \) validly follows from \( X \) (or for short \( X \vDash B \)) iff for every model \( M \), if \( A \) is true in \( M \) for every \( A \in X \), then \( B \) is true in \( M \). In other words, there is no model for which all the premises are True and the conclusion is False.

(Because it makes things prettier, it’s normal to drop set brackets when we are talking about validity. For example, we might say \( A, B \vDash C \) rather than \( \{ A, B \} \vDash C \).)

(e) A set of sentences \( X \) is (semantically) inconsistent iff there is no model in which every \( A \in X \) is true. Otherwise \( X \) is (semantically) consistent.

2. Show these important relationships between the definitions:

(a) \( A \) and \( B \) are equivalent iff \( A \) and \( B \) are true in exactly the same models.

(b) \( A \) is valid iff \( A \) is true in every model.

(c) \( A \) and \( B \) are equivalent iff the sentence \( (A \rightarrow B) \land (B \rightarrow A) \) is valid.

(d) \( A \) is valid iff \( A \) validly follows from the empty set (or \( \vDash A \) for short).

(e) \( A \) and \( B \) are equivalent iff \( A \vDash B \) and \( B \vDash A \).

(f) \( X \vDash \bot \) iff \( X \) is inconsistent.

3. Prove the following, for any sentences \( A \), \( B \), and \( C \):

(a) \( A \land A \equiv A \)

(b) \( (A \land B) \rightarrow A \) is valid

(c) \( (A \rightarrow \bot) \rightarrow \bot \) is equivalent to \( A \)

(d) \( A \rightarrow (B \land C), B \rightarrow \bot \vDash A \rightarrow \bot \)

4. Let \( X \) be a set of sentences, and let \( A, B, \) and \( A_1, \ldots, A_n \) be sentences.
(a) \(X, A \models B \iff X \models A \rightarrow B\)

(b) \(A_1, \ldots, A_n \models B \iff (A_1 \land \ldots \land A_n) \rightarrow B\)

(c) \(X \models A \iff X, A \rightarrow \bot \models \bot\)

5. (Weakening.) For any sets of sentences \(X \subseteq Y\) and any sentence \(A\), if \(X \models A\) then \(Y \models A\).

3.6 Proof

There are lots of different styles of formal proof systems. The oldest is called “Hilbert-style”. A Hilbert-style proof is a sequence of statements, each of which is either an axiom or a premise, or follows from earlier statements by modus ponens. Proof systems like this are especially simple to describe, but a huge pain to use in practice.

Natural deduction style proofs allow us a further kind of step: assumptions. This lets us give proofs that are much easier to follow, and use argument styles like conditional proof, proof by contradiction, and existential instantiation. For instance, here’s a natural-deduction style proof of “if \(p\) then not-not-\(p\)”.

1. Suppose \(p\) (for conditional proof)
2. Suppose \(\neg p\) (for reductio)
3. \(p\) and \(\neg p\) (by 1 and 2, conjunction introduction)
4. \(\neg \neg p\) (by reductio from 2–3, discharging assumption 2)
5. Therefore, \(p \rightarrow \neg \neg p\) (by conditional proof from 1–4, discharging assumption 1)

In some ways this is much more natural than a Hilbert-style proof (hence the name). But the extra complication is that we write lots of things along the way that aren’t logical truths at all, and some of them are even contradictory—like \(p\) and not-\(p\). In order to avoid accidentally “proving” all sorts of ridiculous things in a natural deduction system, it’s very important to keep track at each step what assumptions are in play. This changes through the course of the proof. For instance, when we move to step 3, we are assuming \(p\) and also assuming not-\(p\). When we move to step 5, we are only assuming \(p\); the assumption of not-\(p\) has been “discharged”.
The wordy version written above does, technically, encode all of the information about assumptions, but in a way which is a bit tricky to immediately read off. People have developed lots of different notation systems for doing this assumption bookkeeping more transparently. One is “Fitch-style” notation:

In this version, layers of assumptions are tracked using vertical lines. You can see which assumptions are in play at any particular line of the proof by tracing up each of the lines to its left to the sentence sitting on top of it.

Another notation is “tree-style”:

This version always places statements directly beneath the statements they are directly derived from, and you can trace up the branches of the tree to find the assumptions that are used indirectly. Assumptions that are eventually discharged are marked using brackets and a label (here the numbers 1 and 2), and the step where they’re discharged is marked with a corresponding label.

A third notation is “Gentzen-style” or “sequent-style”.

This is the most verbose version: we explicitly write out all of the assumptions that are used at every step of the proof. Each step of the proof is a sequent, which lists all the assumptions on the left, and sentence we’ve shown can be derived from those assumptions on the right. This is in some ways less convenient for actually working out proofs by hand, because you have to write the same sentences over and
over again. But for the purpose of analyzing proofs, it’s really the best way to go. So that’s the version we’re going to go with officially. (For the purpose of finding proofs, though, feel free to use whatever notation you’re most comfortable with.)

Note that it’s also fairly common to combine the sequent-style idea that assumptions are always explicitly carried through the proof, together with the tree-style idea that steps always go directly beneath the steps they’re immediately derived from. So some people might prefer to write the proof like this:

\[
\begin{align*}
\vdash p &\rightarrow \neg\neg p & \text{conditional proof} \\
\vdash p &\rightarrow \neg\neg p & \text{reductio} \\
\vdash p &\rightarrow p \land \neg p & \text{conjunction intro}
\end{align*}
\]

But for our purposes, since it’s conceptually a little simpler, let’s just think of proofs as lists (where each step follows from steps written somewhere earlier in the proof), rather than trees (where each step follows from its immediate neighbors in the tree).

One more thing. The proof we’ve been looking at shows that \( p \rightarrow \neg\neg p \) is a logical truth: its sequent has no assumptions on the left. But we can also consider proofs where the last line still has assumptions on the left. In that case, the assumptions are premises. What we show in general in a proof is that a certain conclusion validly follows from certain premises. So it’s helpful to read a sequent like \( p \vdash q \rightarrow (q \land p) \) as saying “\( q \rightarrow (q \land p) \) follows from \( p \)”.

If you want some practice with natural deduction proofs, check out this experimental game I’ve been working on!

http://www-bcf.usc.edu/~russ813/logicgame.html

(The premises are at the top, the conclusion is at the bottom. Click on a premise to use its elimination rule, or on the conclusion to use its introduction rule or a proof by contradiction. The buttons at the top are to move between levels or take back moves. Unfortunately the proof system is slightly different from ours, but maybe it will still be helpful to mess around with. It also doesn’t use sequent notation—but it’s a good exercise, and totally mechanical, to translate from natural deduction trees to sequent derivations, just by rewriting all the assumptions that get carried through explicitly at every step.)
3.6 Proof

1. **R.** In this section and the next, let’s fix a countable set $S$ of sentence letters. When I say “sentence”, I mean a sentence in $\text{Prop} \ S$. 

2. **Def.** A sequent is a pair of a set of sentences $X$ and a sentence $A$. We usually write sequents in the notation $X \vdash A$, rather than $(X, A)$. We use the same conventions for writing sets on the left side of $\vdash$ as we use for $\models$. (For example, we label a sequent $A, B \vdash C$ rather than $\{A, B\} \vdash C$. Similarly, if $X$ is a set of sentences, then $X, A \vdash B$ means the same thing as $X \cup \{A\} \vdash B$.)

3. **Def.** If $(s_1, \ldots, s_k)$ is a finite sequence of sequents, then we’ll call each element $s_k$ a line or step. If $i < k$, then we say $s_i$ occurs as an earlier step than $s_k$.

A (formal) proof is a finite sequence of sequents such that each step $X \vdash A$ has (at least) one of the following six properties.

(a) **Assumption.** $A \in X$. (Also called “Trivial”)

(b) **Conjunction Elimination.** For some sentence $B$, either $X \vdash A \land B$ or $X \vdash B \land A$ occurs as an earlier step.

(c) **Conjunction Introduction.** For some sentences $B$ and $C$, $A = B \land C$, and $X \vdash B$ and $X \vdash C$ both occur as earlier steps.

(d) **Conditional Elimination** (Modus Ponens). For some sentence $B$, $X \vdash B \to A$ and $X \vdash B$ both occur as earlier steps.

(e) **Conditional Introduction** (Conditional Proof). For some sentences $B$ and $C$, $A = B \to C$, and $X, B \vdash C$ occurs as an earlier step.

(f) **Contradiction** (Reductio). $X, A \to \bot \vdash \bot$ occurs as an earlier step.

(Remember that $A \to \bot$ is our official way of saying $\neg A$. So the Contradiction rule says: if $X, \neg A$ is inconsistent, then $X \vdash A$.)

Here’s a more concise way of presenting these rules. Read each one as saying, “If you have all the things above the line, then you can derive the thing below the line.”
4. Def.

(a) A sentence $A$ is **provable** from a set of sentences $X$ iff $X \vdash A$ is the last line of some proof. For conciseness, if $A$ is provable from $X$, we also just say $X \vdash A$. (So this notation is ambiguous: $X \vdash A$ can be a name for a certain sequent, or it can be a statement saying that this sequent can be proved. In practice the ambiguity doesn’t usually make trouble.)

(b) Sentences $A$ and $B$ are **provably equivalent** iff $A \vdash B$ and $B \vdash A$. This is abbreviated $A \equiv B$.

(c) A set of sentences $X$ is **(syntactically) inconsistent** iff $X \vdash \bot$. Otherwise $X$ is (syntactically) consistent.

5. Example. Here is a formal proof of $A \rightarrow (A \rightarrow B) \vdash A \rightarrow (A \land B)$. (Note that the comments to the right explain the proof, but aren’t officially part of it.)

\[
\begin{align*}
A \rightarrow (A \rightarrow B), A & \vdash A & \text{Assumption} \\
A \rightarrow (A \rightarrow B), A & \vdash A \rightarrow (A \rightarrow B) & \text{Assumption} \\
A \rightarrow (A \rightarrow B), A & \vdash A \rightarrow B & \rightarrow \text{Elim} \\
A \rightarrow (A \rightarrow B), A & \vdash B & \rightarrow \text{Elim} \\
A \rightarrow (A \rightarrow B), A & \vdash A \land B & \land \text{Intro} \\
A \rightarrow (A \rightarrow B) & \vdash A \rightarrow (A \land B) & \rightarrow \text{Intro}
\end{align*}
\]

(36)

6. Example. Here is a formal proof of $\vdash (A \rightarrow \bot) \rightarrow (A \rightarrow B)$. 

\[
\begin{align*}
& \vdash (A \rightarrow \bot) \rightarrow (A \rightarrow B) \\
\end{align*}
\]
3.6 Proof

\[ A \to \bot, A, B \to \bot \vdash A \]

Assumption

\[ A \to \bot, A, B \to \bot \vdash A \to \bot \]

Assumption

\[ A \to \bot, A, B \to \bot \vdash \bot \]

\[ A \to \bot, A \vdash B \]

Reductio

\[ A \to \bot \vdash A \to B \]

\[ \vdash (A \to \bot) \to (A \to B) \]

Intro

(37)

7. R. General piece of advice for working out proofs: start from the bottom and work upward. That's way easier than trying to do the opposite.

8. Show the following, for any sentences \( A, B, \) and \( C \), by giving explicit formal proofs.

(a) \( A \land A \nvdash A \)

(b) \( \vdash (A \land B) \to A \)

(c) \( (A \to \bot) \to \bot \nvdash A \)

(d) \( A \to (B \land C), B \to \bot \vdash A \to \bot \)

9. R. Note that it immediately follows from the definition that if a sequence of sequents \( (s_1, \ldots, s_{n+1}) \) is a proof, then so is the subsequence \( (s_1, \ldots, s_n) \). (If every step of the longer sequence has one of the six properties, then so does every step of the shorter sequence—since none of the properties “cares” about what comes later in the proof.) This makes it very convenient to prove things by induction on proof length. Say we want to prove that every provable sequent has a certain property \( P \). Then we can reason like this:

Let \( \pi \) be a shortest proof of a sequent that does not have \( P \) (using the Least Number Property). It follows that every step in \( \pi \) earlier than the last step has \( P \). Since \( \pi \) is a proof, we know its last step fits one of the six possible cases. In each case, we can show (using the fact that each earlier step in \( \pi \) has \( P \)) that the last step has \( P \) as well. So the last step of \( \pi \) has \( P \) after all; therefore, every provable sequent has \( P \).

10. Example (Weakening). If \( X \vdash A \), then for any set \( X' \), if \( X \subseteq X' \) then \( X' \vdash A \).

Proof. Suppose that \( X \subseteq X' \), and consider any proof of \( X \vdash A \). We'll prove that \( X' \vdash A \) by induction on the proof length. Suppose (our inductive hypothesis) that if \( Y \vdash B \) is any earlier step in the proof, then if \( Y \subseteq Y' \) then \( Y' \vdash B \). There are six cases to consider.
(a) $A \in X$. In that case, since $X \subseteq X'$, we know $A \in X'$. So $X' \vdash A$ (by Assumption).

(b) For some sentence $B$, the sequent $X \vdash A \land B$ occurs as an earlier step. Then by hypothesis, $X' \vdash A \land B$. So by Conjunction Elimination, $X' \vdash A$. The same reasoning applies in the case where $X \vdash B \land A$ occurs as an earlier step.

(c) For some sentences $B$ and $C$, $A$ is the conjunction $B \land C$, and $X \vdash B$ and $X \vdash C$ occur as earlier steps. In that case, by hypothesis $X' \vdash B$ and $X' \vdash C$. So by Conjunction Introduction, $X' \vdash B \land C$.

(d) For some sentence $B$, $X \vdash B \rightarrow A$ and $X \vdash B$ both occur as earlier steps. Then by hypothesis $X' \vdash B \rightarrow A$ and $X' \vdash B$. So by Conditional Elimination, $X' \vdash A$.

(e) For some sentences $B$ and $C$, $A$ is the conditional $B \rightarrow C$, and $X, B \vdash C$ occurs as an earlier step. Then since $X \cup \{B\} \subseteq X' \cup \{B\}$, by hypothesis $X', B \vdash C$. Then by Conditional Introduction, $X' \vdash B \rightarrow C$.

(f) $X, A \rightarrow \bot \vdash \bot$ occurs as an earlier step. Then $X \cup \{A \rightarrow \bot\} \subseteq X' \cup \{A \rightarrow \bot\}$, so by hypothesis $X', A \rightarrow \bot \vdash \bot$. Then by Contradiction, $X' \vdash A$.

So in any case, $X' \vdash A$.

11. Prove by induction on proof length: if $X \vdash A$, then $X$ has a finite subset $X_0 \subseteq X$ such that $X_0 \vdash A$.

12. Let $X$ be a set of sentences. If every finite subset $X_0 \subseteq X$ is consistent, then $X$ is consistent.

*Hint.* This follows from Exercise 11.

13. (Soundness.) First we prove a few more facts about validity.

(a) If $A \in X$ then $X \vDash A$.

(b) If $X \vDash A \land B$ or $X \vDash B \land A$ then $X \vDash A$.

(c) If $X \vDash B$ and $X \vDash C$ then $X \vDash B \land C$.

(d) If $X \vDash B$ and $X \vDash B \rightarrow A$ then $X \vDash A$.

(e) For any set of sentences $X$ and any sentence $A$, if $X \vdash A$ then $X \vDash A$.

*Hint.* Use induction on proof length. For a couple of the steps, you can refer back to facts about validity we proved in the previous section.

14. (Three versions of Cut.) Let $X$ and $Y$ be sets of sentences and let $B$ be a sentence.
3.7 Completeness

(a) Let $A$ be any sentence. If $X \vdash A$ and $Y, A \vdash B$ then $X, Y \vdash B$.

*Hint.* One way to do this is to use induction on the proof length for $Y, A \vdash B$. Another way is to be clever with a conditional introduction step and a conditional elimination step.

(b) Let $\mathcal{Z}_0$ be a finite set of sentences. Suppose $X \vdash A$ for each $A \in \mathcal{Z}_0$, and suppose $Y, \mathcal{Z}_0 \vdash B$. Then $X, Y \vdash B$.

*Hint.* Use induction on the number of elements of $\mathcal{Z}_0$ to show this stronger fact: if $X \vdash A$ for each $A \in \mathcal{Z}_0$, and $X, Y, \mathcal{Z}_0 \vdash B$ (note the difference) then $X, Y \vdash B$.

(c) Let $\mathcal{Z}$ be any set of sentences. Suppose $X \vdash A$ for each $A \in \mathcal{Z}$, and suppose $Y, \mathcal{Z} \vdash B$. Then $X, Y \vdash B$.

3.7 Completeness

1. **Def.**

(a) A theory is a set of sentences $T$ which is closed under provable consequence. That is, $T$ is a theory iff for any sentence $A$, if $T \vdash A$ then $A \in T$.

(b) We say theory $T'$ extends theory $T$ iff $T \subseteq T'$. The theory $T'$ is a proper extension of $T$ iff $T$ is a proper subset, that is, $T \subsetneq T'$.

(c) A maximal consistent theory is a (syntactically) consistent theory $T$ that has no (syntactically) consistent proper extension. In other words, for any theory $T' \supseteq T$, if $T'$ is consistent then $T' = T$.

2. For any set of sentences $X$, the set $\{ A \in \text{Prop} \mid X \vdash A \}$ is a theory.

*Hint.* Use Cut.

3. For any consistent set of sentences $X$, there is some consistent theory $T \supseteq X$.

4. If $X$ is consistent, then at least one of $X \cup \{ A \}$ and $X \cup \{ A \rightarrow \bot \}$ is consistent.

*Hint.* Prove the contrapositive.

5. **Def.** A set of sentences $X$ is complete (or negation-complete) iff for every sentence $A$, either $A \in X$ or $A \rightarrow \bot \in X$.

6. If $X$ is consistent and complete, then $X$ is a maximal consistent theory.

*Hint.* If $X$ is not a maximal consistent theory, then there is some $A \notin X$ such that $X \cup \{ A \}$ is consistent.
7. Recall that, since $S$ is countable, $\text{Prop} S$ is countable. So there is some one-to-one correspondence $f : \text{Prop} S \to \mathbb{N}$. Let $S_n$ be the set of sentences $A$ such that $fA < n$. Let $X$ be any consistent set of sentences. Prove by induction that for each $n \in \mathbb{N}$, there is a consistent set of sentences $Y_n \supseteq X$ such that for every $A \in S_n$, either $A \in Y_n$ or $A \rightarrow \bot \in Y_n$.

8. (Lindenbaum’s Lemma.) Any consistent theory has a maximal consistent extension. That is, if $T$ is consistent, then there is some maximal consistent theory $T' \supseteq T$.

Hint. If the sets $Y_n$ are constructed as in Exercise 7, then their union

$$Y = \bigcup_n Y_n = \{ A \in \text{Prop} S \mid \text{for some } n \in \mathbb{N}, A \in Y_n \}$$

is consistent and complete (why?) and $X \subseteq T$.

9. Suppose $T$ is a maximal consistent theory, and $A$ is any sentence. The following are equivalent:

(a) $A \in T$
(b) $T \vdash A$
(c) $T, A \nabla \bot$

10. Suppose $T$ is a maximal consistent theory, and $A$ and $B$ are any sentences.

(a) $A \wedge B \in T$ iff $A \in T$ and $B \in T$
(b) If $A \in T$ and $A \rightarrow B \in T$, then $B \in T$.
(c) If $B \in T$, then $A \rightarrow B \in T$.
(d) If $A \notin T$, then $A \rightarrow B \in T$.

11. Let $T$ be a maximal consistent theory. For each sentence letter $s \in S$, let $M_T$ be the truth-assignment

$$M_T(s) = \begin{cases} \top & \text{if } s \in T \\ \bot & \text{if } s \notin T \end{cases}$$

(Note that $M_T$ is the characteristic function of the set $T \subseteq \text{Prop} S$.)

Prove by structural induction that for every sentence $A$,

$$\llbracket A \rrbracket_{M_T} = \top \iff A \in T$$


In short, then, every maximal consistent theory has a model.

12. (Henkin’s Lemma.) Any (syntactically) consistent theory has a model. That is, if \( T \) is consistent, then there is some model \( M \) in which every sentence \( A \in T \) is true.

   *Hint.* Use the previous fact along with Lindenbaum’s Lemma.

13. (Completeness.) For any set of sentences \( X \) and any sentence \( A \), if \( X \models A \) then \( X \vdash A \).

14. (Compactness.) For any set of sentences \( X \) and any sentence \( A \), if \( X \models A \), then there is some finite subset \( X_0 \subseteq X \) such that \( X_0 \not\models A \).

   *Hint.* Remember 3.6.11.

15. For any set of sentences \( X \), if every finite subset \( X_0 \subseteq X \) has a model, then \( X \) has a model.

4 Terms

4.1 Syntax

We’re going to work our way up to first-order logic with identity and function symbols (“predicate logic” for short). But there’s a useful resting place intermediate between propositional logic and predicate logic, which generalizes some of the ideas we’ve already been working with. Sentences in propositional logic denote truth functions—functions from truth-values to truth-values, and more generally from sequences of truth-values to truth-values. But more generally, we’ll want to consider expressions that denote other sorts of functions as well—like the successor and addition functions for natural numbers. So in this part, we’ll work out how “term languages” for denoting functions work in general. All of this is a straightforward generalization of what we already did with truth functions, so it’s just a matter of getting used to a bit of extra abstraction. After that, we’ll go on to the next step of adding in quantifiers.

In this part we’re not just describing one language, but a kind of language, which has different specific instances. One specific instance is the language of truth functions. Another will be the language of arithmetical functions. And there will be others as well. These languages are distinguished by their inventory of primitive function symbols, which are our basic building blocks for building up terms. For instance, the primitive
function symbols of $\text{Prop}$ are $\bot$, $(\land)$, and $(\rightarrow)$. Note that these primitives don’t all behave precisely the same way. The symbol $\bot$ can be used to make a term all on its own, while the symbols $(\land)$ and $(\rightarrow)$ each need two input terms to build up a new one. We say that $(\land)$ is a binary connective (two arguments), while $\bot$ is a nullary connective (zero arguments). The negation symbol $(\neg)$ would be a unary connective (one argument). In general, a term language has a stock of $n$-ary function symbols. If $f$ is an $n$-ary function symbol, then the number $n$ is called its “arity”.

Here’s the plan. A “signature” is a description of the primitive function symbols (or “operations”) for a term language. Given any signature, we can define the language generated by that signature: that is, arbitrary terms that are built up by applying these operations to variables. In this section we’ll precisely describe the syntactic structure of terms, and an important way of “sticking together” terms using substitution. After that, we’ll describe how to interpret terms as denoting functions. Then we’ll consider the theory of identity for terms like this, and describe a sound and complete proof system for identity. This will put in place most of the pieces we’ll need for predicate logic with identity. (The main thing left out, which we won’t cover until the next part, is quantifiers.)

1. **Def.** A **signature** $L$ consists of a set of **function symbols** and an **arity** function that assigns each function symbol $f$ a natural number $\text{ar}(f)$. If $\text{ar}(f) = n$, then $f$ is an $n$-ary function symbol (also called an $n$-place operation). The special case of a 0-ary function symbol is called a **name**.

2. **Example.** The **signature of the language of Boolean functions** consists of a name $\bot$, a two-place function symbol $(\land)$, and a two-place function symbol $(\rightarrow)$. That is,

$$
\begin{align*}
\text{ar}(\bot) &= 0 \\
\text{ar}(\land) &= 2 \\
\text{ar}(\rightarrow) &= 2
\end{align*}
$$

(41)

3. **Example.** The **signature of the language of arithmetic** consists of a name $0$, a one-place function symbol $\text{suc}$, a two-place function symbol $(+)$, and a two-place function symbol $(\cdot)$.

$$
\begin{align*}
\text{ar}(0) &= 0 \\
\text{ar}(\text{suc}) &= 1 \\
\text{ar}(+) &= 2 \\
\text{ar}(\cdot) &= 2
\end{align*}
$$

(42)

4. **Unofficial Def.** Let $L$ be a signature, and let $V$ be a set of **variables**. The set of $L$-**terms** with variables in $V$, $\text{Term}_L V$, is recursively defined as follows:

(a) Every variable $v \in V$ is an element of $\text{Term}_L V$. 

4. **TERMS**
For every \( n \)-ary function symbol \( f \), and any sequence of \( n \) terms \( a \in \text{Term}_L V \), then \( fa \) is an element of \( \text{Term}_L V \). (That is, \( f(a_1, \ldots, a_n) \) is an element of \( \text{Term}_L V \).

(In the special case where \( n = 0 \), that is, \( f \) is a name, we don’t bother to write down its empty sequence of arguments. That is, for a name \( a \), we write the term as just \( a \) rather than \( a() \).

(c) Nothing is an element of \( \text{Term}_L V \) unless it can be constructed by applying steps (a) and (b) finitely many times.

As usual for inductive structures, we can state this more officially in terms of an inductive property, just as we did for natural numbers and for the sentences of propositional logic. This means, as before, that we can do proof by induction on the structure of terms. It also means that, as before, we can recursively define functions that take each term to some value.

6. Official Def. The \( L \)-terms with variables in \( V \) are a set \( \text{Term}_L V \) such that

\[ V \subseteq \text{Term}_L V \] (each variable is a term), and for each \( n \)-ary function symbol \( f \) in \( L \) there is a corresponding term-forming function \( f : (\text{Term}_L V)^n \rightarrow \text{Term}_L V \). \( f \) takes a sequence of \( n \) terms to a term. We don’t bother to notationally distinguish the function symbol itself from its corresponding term-forming operation.) Furthermore:

(a) (Injective Property) For each \( f \), the term-forming function is one-to-one, and no variable is in its range. For any distinct \( f \) and \( g \) the ranges of their term-forming functions have no elements in common.

(b) (Inductive Property) Let \( S \subseteq \text{Term}_L V \). Suppose \( v \in S \) for each variable \( v \in V \), and suppose that for each \( n \)-ary function symbol \( f \), whenever \( a \in S^n \) then \( fa \in S \). Then \( S = \text{Term}_L V \).

7. Example. Consider a signature \( L \) with a two-place function symbol \( f \), and let \( V = \{x, y\} \). Then these are examples of elements of \( \text{Term}_L V \):

- \( x \)
- \( f(x, x) \)
- \( f(x, f(x, y)) \)
- \( f(f(f(y, x), x), x), y) \)

8. Example. These are examples of terms in the language of arithmetic, with variables in \( \{x, y\} \). (Note that as usual we write, e.g., \( x + y \) rather than \( (+)(x, y) \).)
9. **Def.**

(a) In the language of arithmetic, let 1 be suc 0, let 2 be suc 1, etc.

(b) For each natural number \( n \in \mathbb{N} \), the **numeral** for \( n \) is a term in the language of arithmetic, defined recursively:

i. The numeral for the number 0 is the name 0.

ii. If the numeral for \( n \) is \( a \), then the numeral for the number \( n + 1 \) is the term \( \text{suc} \ a \).

10. **Example.** The terms of the language of Boolean functions are the very same thing as sentences of the propositional language. That is, for the signature of the language of Boolean functions, the terms in \( \text{Term}_L V \) are just the same as the sentences in \( \text{Prop} V \).

11. Prove by induction that every element of \( \text{Term}_L V \) is either a variable or else \( fa \) for some \( n \)-place function symbol \( f \) and some sequence of terms \( a \in (\text{Term}_L V)^n \).

   **Hint.** Remember 2.1.7 and 3.1.8.

12. Let \( L \) be any signature, let \( V \) be any set of variables, and let \( T = \text{Term}_L V \).

   (a) Give a recursive definition of the set of variables that occur in a term \( a \in T \). Here are some examples (for the language of arithmetic),

   - The set of variables that occur in \( x \) is \( \{x\} \).
   - The set of variables that occur in \( x + (\text{suc} \ x \cdot (y + x)) \cdot 0 \) is \( \{x,y\} \).

   (b) Show by induction that, for any term \( a \in T \), the set of variables that occur in \( a \) is finite.

   (c) Suppose that \( L \) has just a one-place function symbol \( f \) and no other function symbols. Show by induction that for every \( a \in \text{Term}_L V \), exactly one variable occurs in \( a \).

13. **Def.**
(a) A signature $L$ is **finite** iff $L$ has finitely many primitive operations. $L$ is **countable** iff $L$ has countably many primitive operations.

(b) A signature $L^+$ **extends** a signature $L$ iff it includes all the operations from $L$ (with the same arities) and possibly adds in additional operations. Note that in this case, each $L$-term can be identified with a syntactically identical term in the extended language of $L^+$.

14. For any $L$-term $a$, there is a finite signature $L_0$ such that $L$ extends $L_0$, and $a$ is (syntactically identical to) an $L_0$-term.

15. Let $L$ be any signature, let $V$ be any set of variables, and let $T = \text{Term}_L V$.

   (a) Give a recursive definition of the **complexity** of a term in $T$, which intuitively is the number of function symbols that appear in it. Here are some examples (from the language of arithmetic).

   - The complexity of $x$ is 0.
   - The complexity of $y + 2$ is 4. (Remember that $2 = \text{suc suc 0}$.)
   - The complexity of $x + \text{suc } x \cdot (y + x) \cdot 0$ is 6.

   (b) Prove the **least complexity principle** for terms: if $S \subseteq T$ is a non-empty set, then $S$ has an element $a \in S$ such that the complexity of $a$ is less than or equal to the complexity of any element of $S$.

16. (This problem generalizes 3.1.14.) Suppose $L$ is a countable signature. Suppose also that $V$ is countable. Let $T = \text{Term}_L V$, and let $T_n$ be the set of elements of $T$ with complexity at most $n$.

   (a) Let $f$ be a $k$-place operation, and let $fT$ be the set of terms of the form $fa$ for some $a \in T^k$. (That is, the terms in $fT$ are of the form $f(a_1, \ldots, a_k)$ for a sequence of $k$ terms $a_1, \ldots, a_k$.) Then $fT \cap T_n$ is the set of terms of that form, with complexity at most $n$. Define a one-to-one function from $fT \cap T_n$ to $T_{n-1}^k$.

   (b) For each $n$, $T_n$ is countable.

   **Hint.** Use induction to show there is a countable partition of $T_n$ into countable sets.

   (c) $T$ is countable.

17. **R.** As with propositions, since it doesn’t really matter what particular things we choose to use as variables, it can be convenient to use numbers. Again, since it looks a bit weird and confusing to insert numbers into our formulas, we’ll...
adopt the convention of using $x_1, \ldots, x_n$ as labels for the elements of $n$; or if $n$ is a small number then we use $x, y, z$. So $\text{Term}_L n$ (that is, $\text{Term}_L n$) is the set of $L$-terms with at most $n$ variables. In fact, from here on out we’ll only be considering terms with a finite alphabet of variables.

18. **Def.** Consider any term $a \in \text{Term}_L n$, and any sequence of terms $b \in (\text{Term}_L V)^n$. (In other words, $b_1, \ldots, b_n$ are each terms in $\text{Term}_L V$.) Then the **substitution instance** $a[b]$ (or if you prefer, $a[b_1, \ldots, b_n]$) is defined recursively as follows:

(a) For any variable $x_i$,

$$ x_i[b] = b_i \tag{43} $$

(b) For any $k$-place function symbol $f$, and any sequence of terms $a \in (\text{Term}_L n)^k$,

$$ (fa)[b] = f(a_1[b], \ldots, a_k[b]) \tag{44} $$

19. **Example.** In a language with a name $a$, a one-place function symbol $g$, and a two-place function symbol $f$:

$$ (f(x, a))[gy] = f(gy, a) $$
$$ x[(f(y, z)] = f(y, z) $$
$$ (f(f(a, g)), x)[f(y, a)] = f(f(a, f(y, a)), f(y, a)) \tag{45} $$

In the language of arithmetic:

$$ (x + y)[0, x \cdot x] = 0 + x \cdot x $$
$$ ((0 + x \cdot y) + x)[x + x, 0 \cdot x] = (0 + (x + x) \cdot (0 \cdot x)) + (x + x) \tag{46} $$

20. Show the following properties of substitution (using induction):

(a) For any term $a \in \text{Term}_L n$,

$$ a[x_1, \ldots, x_n] = a \tag{47} $$

(b) For any terms $a \in \text{Term}_L 1$, $b \in \text{Term}_L n$, and any sequence of terms $c \in (\text{Term}_L V)^n$.

$$ a[b][c] = a[b[c]] \tag{48} $$
In other words,

\[ a[b][c_1, \ldots, c_n] = a[b[c_1, \ldots, c_n]] \quad (49) \]

\[(c) \quad \text{For any term } a \in \text{Term}_L m, \text{ and any sequences of terms } b \in (\text{Term}_L n)^m \text{ and } c \in (\text{Term}_L V)^n, \]

\[ a[b][c] = a[b_1[c], \ldots, b_m[c]] \quad (50) \]

In other words,

\[ a[b_1, \ldots, b_m][c_1, \ldots, c_n] = a[b_1[c_1, \ldots, c_n], \ldots, b_m[c_1, \ldots, c_n]] \quad (51) \]

### 4.2 Structures and Interpretations

We’re thinking of each term in our language as representing some function. Let’s spell this idea out—in a way that is a direct generalization of what we did in the special case of truth functions. One extra complication, in comparison to truth-functions, is that we have to specify the domain of the functions, and the meanings of the primitive function symbols. (For propositional sentences we just always used the same domain, \( \mathbb{2} \), and the same interpretations of the basic operations like conjunction.) We specify these using what’s called a structure. We’ll interpret terms within some structure or other. Intuitively you can think of a structure as something like a “possible world”.

In this section we’ll describe how to interpret terms in a structure, and then we’ll show a few things about relationships between different structures. The most important relationship is isomorphism: two isomorphic structures are “essentially the same”, in that one of them basically just amounts to relabeling the domain of the other. The idea of an isomorphism extends the idea of a one-to-one correspondence between sets, to a mapping that doesn’t just preserve cardinality, but also preserves structure.

1. **Def.** Let \( D \) be a set: the **domain**.

   (a) An **\( n \)-place operation** on \( D \) is a \( D^n \to D \) function. Let \( \text{Op}_D n \) be the set of \( n \)-place operations on \( D \). That is, \( \text{Op}_D n = D^n \to D \).

   (b) An **\( (n \)-place) assignment** in \( D \) is an element of \( D^n \). So a function in \( \text{Op}_D \) takes each assignment to an element of \( D \).
2. **Example.** $\text{Bool } n = \text{Op}_2 n$.

3. **Def.** Let $L$ be a signature. An $L$-structure $S$ is a domain $D_S$ (we often drop the subscript) together with, for each $n$-ary function symbol $f$ in $L$, an $n$-place operation $f_S \in \text{Op}_{D_S} n$, which is called the extension of $f$ in $S$.

(L-structures are also sometimes called models.)

4. **R.** Note that in the special case where $a$ is a name (a zero-place function), then $a_S$ is technically a zero-place operation, which is a function $D^0 \to D$; that is, a function that takes the empty sequence () to some element in $D$. But as we know, we can just think of a zero-place operation as a way of picking out an element of $D$. (Remember the case of $\text{Bool } 0$.) So we’ll often just think of the extension of a name $a_S$ as an element of the domain $D$.

5. **Example.** The **standard Boolean model** is a structure for the language of Boolean functions which we’ll just call $\mathcal{2}$ (despite the fact that this symbol has several other meanings). The domain $D_2 = \mathcal{2}$ (the two-element set) and the extension for each function symbol is its corresponding truth function:

\[
\begin{align*}
\bot_2 &= \bot \\
\land_2 &= \land \\
\rightarrow_2 &= \rightarrow 
\end{align*}
\]  

(52)

6. **Example.** The **standard model of arithmetic** is a structure for the language of arithmetic, which we’ll just call $\mathbb{N}$. The domain is $\mathbb{N}$ (the set of natural numbers), and the extensions are just what you would expect:

\[
\begin{align*}
0_{\mathbb{N}} &= 0 \\
\text{suc}_{\mathbb{N}} &= \text{suc} \\
(+)_{\mathbb{N}} &= (+) \\
(\cdot)_{\mathbb{N}} &= (\cdot)
\end{align*}
\]  

(53)

7. **Example.** The **trivial model** $\mathcal{1}_L$ for an arbitrary signature $L$ has domain 1 the set with just one element, and takes every $n$-place operation $f$ to the only function $f_1 : 1^n \to 1$. (That is, $f_1$ is the function which takes $\bullet \in 1$ as its output for every input).

8. **Def.** Let $S$ be an $L$-structure. The **interpretation** in $S$ gives us (for each $n \in \mathbb{N}$) a function

\[
\llbracket \cdot \rrbracket_S : \text{Term}_L n \to \text{Op}_{D_S} n
\]  

This is defined recursively as follows, for any assignment $d = (d_1, \ldots, d_n) \in D^n$.

(It’s normal to write $\llbracket a \rrbracket_S^d$, or sometimes $\llbracket a \rrbracket_S d$, rather than $\llbracket a \rrbracket_S d$. But it still just means function application.)
4.2 Structures and Interpretations

(a) For a variable \( x_i \),

\[
\|x_i\|_d^S = d(x_i) = d_i
\]  \hspace{1cm} (55)

(b) For a \( k \)-place operation \( f \) and a sequence of \( k \) terms \( a = (a_1, \ldots, a_k) \in (\text{Term}_L)_n^k \),

\[
\|f(a)\|_d^S = f_\delta(\|a_1\|_d^S, \ldots, \|a_k\|_d^S)
\]  \hspace{1cm} (56)

9. **Example.** The interpretation function \( \|\cdot\|_2 \), using the standard Boolean model \( \mathbb{B} \) as our structure, is exactly the same as the interpretation function for the propositional language that we defined earlier.

(One small notational point: I guess in Part 3 I put the truth-assignment as a subscript, but now it would show up as a superscript—our earlier \( M \) has turned into our current \( d \). Later I’ll go back and make the notation more consistent.)

10. Show the following from the definitions, using the standard model of arithmetic \( \mathbb{N} \).

(a) \( \|0 + 0 \cdot x\|_{\mathbb{N}}^{(m,n)} = 0 \) (for any \( m, n \in \mathbb{N} \))

(b) \( \|(x + y) + x\|_{\mathbb{N}}^{(m,n)} = \|x + (x + y)\|_{\mathbb{N}}^{(n,m)} \) (for any \( m, n \in \mathbb{N} \))

(c) \( \|x \cdot 0 + y\|_{\mathbb{N}} = \|y\|_{\mathbb{N}} \)

11. Suppose \( m \) and \( n \) are natural numbers, and \( a \) and \( b \) are their respective numerals (Def 4.1.9). Let \( c \) be the numeral for the number \( m + n \). Show that \( \|a + b\|_{\mathbb{N}} = \|c\|_{\mathbb{N}} \).

12. **Def.** We can make things prettier in a lot of places by adopting a couple extra bits of notation for dealing with sequences of functions and sequences of terms. First, we can use “general function composition” (which was also defined earlier in 3.4.7). If \( f : B^k \to C \) is an \( n \)-place function, and \( g = (g_1, \ldots, g_n) \in (A \to B)^n \) is a sequence of \( n \) functions, then the function \( (f \ast g) : A \to C \) is defined thus:

\[
(f \ast g)a = f(g_1a, \ldots, g_na) \quad \text{for any } a \in A
\]  \hspace{1cm} (57)

We can also extend our use of the interpretation function \( \|\cdot\| \) to apply to sequences of terms.
\[(a_1, \ldots, a_n) \overset{S}{\longrightarrow} (\langle a_1 \rangle_S, \ldots, \langle a_n \rangle_S)\] (58)

Using these two shorthands we can often write things much more concisely and keep a lot of the sequence-juggling in the background. For instance, clause (b) of the definition of \(\llbracket \cdot \rrbracket\) can instead be written:

\[\llbracket \phi \rrbracket_S = f_S \ast \llbracket \phi \rrbracket_S\] (59)

I’ll try not to lean too heavily on this notational trick, in case you find the extra abstraction confusing, but if you can get used to it you might find it simplifies things quite a bit.

13. * Consider a term \(a \in \text{Term}_L n\) and a sequence of terms \(b \in (\text{Term}_L V)^n\). Let \(S\) be any \(L\)-structure.

(a) Show by induction that, for any assignment \(d \in D^V\),

\[\llbracket a[b] \rrbracket_S = \llbracket a \rrbracket_S(\llbracket b_1 \rrbracket_S, \ldots, \llbracket b_n \rrbracket_S)\] (60)

In other words, using the more concise notation for sequences:

\[\llbracket a[b] \rrbracket_S = \llbracket a \rrbracket_S \ast \llbracket b \rrbracket_S\] (61)

(b) (Substitution of Equivalents; i.e. Leibniz’s Law.) Let \(S\) be an \(L\)-structure, let \(a\) and \(a'\) be terms in \(\text{Term}_L n\), and let \(b\) and \(b'\) be sequences of terms in \((\text{Term}_L V)^n\). Suppose that \(\llbracket a \rrbracket_S = \llbracket a' \rrbracket_S\) and also \(\llbracket b_i \rrbracket_S = \llbracket b'_i \rrbracket_S\) for each \(i \in \mathbb{N}\). Then

\[\llbracket a[b] \rrbracket_S = \llbracket a'[b'] \rrbracket_S\] (62)

14. Suppose \(L^+\) extends \(L\), and let \(S^+\) be an \(L^+\) structure. Then there is an \(L\)-structure \(S\) such that, for any term \(a\) in the smaller language \(L\), \(\llbracket a \rrbracket_S = \llbracket a \rrbracket_{S^+}\). Call this the reduction of \(S^+\) to \(L\).

15. **Def.** An isomorphism of \(L\)-structures \(S\) and \(T\) is a one-to-one correspondence \(h : D_S \rightarrow D_T\) such that, for every \(n\)-ary operation \(f_i\) and any \(a \in D_S^n\),

\[h(f_S(a_1, \ldots, a_n)) = f_T(ha_1, \ldots, ha_n)\] (63)

Structures \(S\) and \(T\) are isomorphic (written \(S \cong T\)) iff there exists an isomorphism from \(S\) to \(T\).
4.3 Identity

16. **Example.** Consider the empty signature $L_0$, which has no function symbols. An $L_0$-structure is just given by its domain—a set. Then for $L_0$ structures $S$ and $T$, an isomorphism is just a one-to-one correspondence between $D_S$ and $D_T$. So the isomorphism definition of $\cong$ generalizes the earlier one-to-one correspondence definition of $\cong$ for sets.

17. Give a structure of the language of arithmetic which is not isomorphic to the standard model of arithmetic $\mathbb{N}$.

18. Consider the following structure $S$ for the language of arithmetic. The domain $D_S$ is the set of even numbers.

$$0_S = 0 \quad \text{suc}_S = \text{suc} \cdot \text{suc} \quad (+_S) = (+) \quad m \cdot_S n = (m \cdot n)/2$$

Show that $S$ is isomorphic to the standard model of arithmetic $\mathbb{N}$.

(We haven’t given an official definition of division, but just take for granted that $(m \cdot n)/2$ is a number $k$ such that $2 \cdot k = m \cdot n$.)

19. Suppose $h : S \to T$ is an isomorphism. If $d$ is an assignment in $D_S^n$, then $h \circ d$ is an assignment in $D_T^n$, where $(h \circ d)_i = h(d_i)$ for each $i \in n$. Show by induction that for every term $a \in \text{Term}_L n$,

$$h(\lceil a \rceil_S) = \lceil a \rceil_{T}$$

In particular, $n = 0$, so $a$ is a term with no variables, then

$$h(\lceil a \rceil_S) = \lceil a \rceil_T$$

20. For any structures $S$, $T$, and $U$:

(a) $S \cong S$

(b) If $S \cong T$ then $T \cong S$.

(c) If $S \cong T$ and $T \cong U$ then $S \cong U$.

4.3 Identity

Intuitively, a term language gives us a pretty rich system of names for things, but it doesn’t give us a way to say anything about those things. As we continue to work up toward first-order logic, let’s first look at a much simpler language. This simple
language only includes simple identity statements, which say that \(a\) is precisely the same thing as \(b\). (It doesn't even include statements like “if \(a = b\) then \(c = d\).”) In this section we’ll introduce a simple proof theory for this language, and prove that it’s sound and complete with respect to our interpretations. This will make up an important piece of our soundness and completeness theorem for first-order logic.

For this section, we’ll only be looking at closed terms, which do not contain any variables: they are built entirely from names and function symbols. So we’ll assume the set of variables is empty. (Note that in this case, if the signature \(L\) doesn’t include any names, then there are no \(L\)-terms at all.)

1. **Def.** For this section, fix a signature \(L\), and let \(T = \text{Term}_L 0\). A **closed term** is an element \(T\). A **term of one variable** is an element of \(\text{Term}_L 1\).

2. **Def.** An **identity formula** is a pair of terms \((a, b)\) from \(\text{Term}_L n\). We write identity formulas as \((a = b)\). An **identity sentence** (or just an **identity**) is an identity formula where \(a\) and \(b\) are closed terms.

3. **R.** Sometimes people use a different symbol for identity formulas (such as \(\doteq\) or \(\equiv\) or \(\sim\)), in order to distinguish these formal identities we are describing from genuine identity statements that we want to assert. I won’t use a different symbol, but this means we do need to be extra careful to avoid ambiguity when we talk about identities. For instance, rather than saying something like “\((a = b) = (a' = b')\),” it’s better to say something like “The identity \((a = b)\) is the same as the identity \((a' = b')\).”

When there are no variables to worry about, rather than writing \(\llbracket a \rrbracket_S\) all the time, we’ll just drop the empty assignment out of our notation. So \(\llbracket a \rrbracket_S\) is an element of the domain \(D_S\).

4. **Def.**

   (a) For closed terms \(a\) and \(b\) and an \(L\)-structure \(S\), we say \((a = b)\) is **true in** \(S\) iff \(\llbracket a \rrbracket_S = \llbracket b \rrbracket_S\). Otherwise \((a = b)\) is **false** in \(S\).

   (b) For a set of identities \(X\) and an identity \(A\), we say \(A\) **validly follows from** \(X\) (for short, \(X \vDash A\)) iff there is no structure \(S\) in which each element of \(X\) is true, and \(A\) is false.

   (c) An identity \(A\) is **valid** iff \(A\) validly follows from the empty set.

5. **Example.** Consider the signature \(L\) with a two-place function symbol \(f\) and names \(a\) and \(b\).
4.3 Identity

(a) \( f(a, b) = f(a, b) \) is valid, since this is true in every structure. That is, for any \( L \)-structure \( S \), it’s obvious that \( \llbracket f(a, b) \rrbracket_S \) is the same as \( \llbracket (a, b) \rrbracket_S \).

(b) \( f(a, b) = f(b, a) \) is not valid, since there are \( L \)-structures in which this is false. For a simple example (there are many), consider the structure \( S \) where the domain is \( \{0, 1\} \), and \( f \) is interpreted \( f_S(d_1, d_2) = d_1 \) for any pair \( (d_1, d_2) \in D_S^2 \), and the names \( a \) and \( b \) are interpreted so \( a_S = 0 \), and \( b_S = 1 \). Then

\[
\llbracket (a, b) \rrbracket_S = f_S(a, b_S) = f_S(0, 1) = 0
\]
\[
\llbracket (b, a) \rrbracket_S = f_S(b, a_S) = f_S(1, 0) = 1
\]

Obviously these are different. So \( f(a, b) = f(b, a) \) is false in \( S \).

6. Suppose \( L \) is the language of arithmetic, extended with two new names \( m \) and \( n \). Use the definitions to check which of the following are true. (Remember that 1 means \( \text{suc} 0 \), and 2 means \( \text{suc} 1 \).)

(a) \( m = n + 1 \models 1 + (1 + m) = 1 + (1 + (n + 1)) \)

(b) \( m = n \models n = m \)

(c) \( \models m + n = n + m \)

(d) \( m = n + n \models 1 \cdot m = 2 \cdot n \)

7. Let \( S \) and \( T \) be \( L \)-structures, and let \( A \) be an identity sentence. Suppose \( S \cong T \). If \( A \) is any identity, then \( A \) is true in \( S \) iff \( A \) is true in \( T \).

8. **Def.** We can define substitution for identities in an obvious way: substituting a term into an identity just means substituting it into each of the two terms in the identity. Let \( A[x] \) be an identity formula of one variable; that is, it’s a formula \( (a[x] = b[x]) \) for some terms \( a[x] \) and \( b[x] \) of one variable. Let \( e \) be a closed term. Then the **substitution instance** \( A[e] \) is the identity sentence \( (a[e] = b[e]) \).

9. Prove the following:

(a) (Identity.) For any term \( a \) and any set of identities \( X \),

\[ X \models a = a \] (68)

(b) (Substitution.) For any identity \( A[x] \) with one variable, and any closed terms \( e \) and \( e' \), if \( X \models e = e' \), and \( X \models A[e] \), then \( X \models A[e'] \).

**Hint.** Remember 4.2.19.
10. **Def.** A **sequent** is a pair of a set of identities $X$ and an identity $A$, written $X \vdash A$. To be more explicit about the language $L$ of $X$ and $A$, we’ll also sometimes write $X \vdash_{L} A$.

11. **Def.** A **proof** is a finite sequence of sequents such that for each step $X \vdash A$, one of the following holds:

(a) **Assumption.** $A \in X$.

(b) **Identity.** $A$ is $(a = a)$, for some term $a$.

(c) **Substitution** (Leibniz’s Law). For some identity of one variable $B[x]$, and some closed terms $c$ and $c'$, $A$ is $B[c']$, and $X \vdash c = c'$ and $X \vdash B[c]$ both occur as earlier steps. (In other words, for some terms of one variable $a$ and $b$, $A$ is $(a[c'] = b[c'])$, and $X \vdash c = c'$ and $X \vdash a[c] = b[c]$ both occur earlier. Intuitively this says that if we know $c$ and $c'$ are identical, then anything we know to be true of $c$ we can also infer to be true of $c'$.)

We can concisely sum these up with these inference diagrams:

\[
\frac{X, A \vdash A}{X \vdash A} \quad \text{Assumption} \quad \frac{X \vdash a = a}{X \vdash A[a]} \quad \text{Identity} \quad \frac{X \vdash a = b}{X \vdash A[b]} \quad \text{Substitution}
\]

12. **Def.** We say $X \vdash A$ is **provable** iff this sequent occurs as the last line of some proof. (As usual, we often just say $X \vdash A$ as a shorthand for the claim that the sequent is provable.)

13. **Example.** Say $f$ is a two-place function symbol, and $a$ and $b$ are names. We’ll prove $a = b \vdash f(a, a) = f(b, a)$.

\[
\begin{align*}
  a & = b \vdash a = b & \text{Assumption} \\
  a & = b \vdash f(a, a) = f(a, a) & \text{Identity} \\
  a & = b \vdash f(a, a) = f(b, a) & \text{Substitution}
\end{align*}
\]

For the substitution step, the identity formula of one variable $B[x]$ is $f(a, a) = f(x, a)$. When we substitute $b$ into this, we get $f(a, a) = f(b, a)$, and when we substitute $a$ into this, we get the earlier step $f(a, a) = f(a, a)$.

14. Show the following (for arbitrary terms $a$, $b$, $c$, and $d$, and where $f$ is a two-place function symbol):
4.3 Identity

(a) \( a = b \vdash f(a, c) = f(b, c) \)

(b) \( a = c, b = d \vdash f(a, b) = f(c, d) \)

(c) (Symmetry) \( a = b \vdash b = a \)

(d) (Euclidean Property) \( a = c, b = c \vdash a = b \)

(e) (Transitivity) \( a = b, b = c \vdash a = c \)

15. Let \( f \) be an \( n \)-place function symbol. Suppose

\[
X \vdash a_1 = b_1 \quad \ldots \quad X \vdash a_n = b_n
\]  

(71)

In that case, if \( X \vdash f(a_1, \ldots, a_n) = c \), then \( X \vdash f(b_1, \ldots, b_n) = c \).

16. (Soundness.) If \( X \vdash A \) then \( X \models A \).

Hint. As we did for propositional logic, use induction on proof length.

17. (Henkin’s Lemma.) Let \( X \) be any set of identities.

Our goal is to come up with a structure in which every identity in \( X \) is true. Our strategy is to use the terms in \( T \) themselves as the raw materials for this structure. But of course, \( X \) might include identities like \( f(a, b) = ga \), while these are clearly distinct terms. So we need to “collapse” some of these different terms onto each other. This is a pretty common problem—we’re distinguishing too many different things of the same kind, and we want to blur them all together into one thing of each kind. It also has a standard solution. Instead of the terms themselves, we’ll instead use certain sets of terms as the elements of our structure’s domain: sets called “equivalence classes”.

For any \( L \)-term \( a \), let \( \bar{a} \) be the set of terms provably equal to \( a \), from premises in \( X \).

\[
\bar{a} = \{ a' \in T \mid X \vdash a = a' \}
\]  

(72)

Call this the equivalence class for term \( a \). We’ll define a certain structure \( S \) as follows. Let the domain be the set of all equivalence classes.

\[
D_S = \{ \bar{a} \mid a \in T \}
\]  

(73)

For each \( n \)-ary operation \( f \), let

\[
f_S(\bar{a}_1, \ldots, \bar{a}_n) = f(a_1, \ldots, a_n)
\]  

(74)
(a) Check that this function $f_S$ is well-defined. That is, if two sequences of terms $a$ and $b$ have the very same equivalence classes $\overline{a_i} = \overline{b_i}$ for each $i$, then $\overline{fa} = \overline{fb}$.

(b) Show by induction that for any term $a \in \text{Term}_n$, and any sequence of terms $b$,

$$\|e\|_S(\overline{b_1}, \ldots, \overline{b_n}) = \overline{a[b]}$$  \hfill (75)

In particular, for a closed term $a$,

$$\|a\|_S = \overline{a}$$  \hfill (76)

(c) For any terms $a$ and $b$, $X \vdash a = b$ iff $\overline{a} = \overline{b}$.

(d) Conclude that for any set of identities $X$, there is a structure $S$ such that if $A$ is any identity sentence, then $A$ is true in $S$ iff $X \vdash A$.

18. (Completeness.) If $X \not\models A$ then $X \not\vdash A$.

5 Quantifiers

This is the Captain Planet point of the story, when all our powers combine. We’ve arrived at first-order logic with function symbols and identity (“predicate logic”) which combines all the resources of propositional logic, functional terms, and identity statements, and then adds in one more ingredient: quantifiers.

Note there’s one resource that it’s normal to include in first-order logic, which we’re leaving out: primitive predicates (besides identity). (So “predicate logic” is bit of a misnomer.) As it turns out, we won’t need to do anything with those, and it makes our life a little simpler to just ignore them.

(Also, adding predicates in wouldn’t make much of a difference. Since we do have function symbols, names, and identity, we can use a trick of Frege’s. We can introduce a name “True”; then instead of using a predicate like “loves(x, y)” we can instead use a function symbol, and say “the lovingness of (x, y) = True”. So predicates don’t really add special expressive power.)
5.1 Syntax

Say for example our language $L$ has a name $a$ and a two-place function symbol $f$.

Formulas of predicate logic are built up in three ways. Some formulas are identities. These are all formulas with at most two free variables ($x$ and $y$).

$$(x = y) \quad (f(x, a) = x) \quad (a = a) \quad (y = f(a, f(a, y)))$$  \hspace{1cm} (77)

We can also build up formulas using arbitrarily complicated propositional formulas. So these are formulas (of at most two free variables) as well.

$$(x = y) \rightarrow (x = x) \quad \bot \quad (f(a, x) = f(y, y)) \land ((a = a) \rightarrow \bot)$$  \hspace{1cm} (78)

(We can get these by substituting identity formulas into the propositional formulas $p \rightarrow q$, $\bot$, and $p \land (q \rightarrow \bot)$, respectively.)

We can also build up formulas using quantifiers.

$$\forall x(x = x) \quad \forall x \bot \quad \forall x \forall y((f(a, x) = f(y, y)) \rightarrow \bot)$$  \hspace{1cm} (79)

An important thing about quantifiers is that they “bind” variables. This means that there are fewer free variables available “outside” of a quantifier than “inside” it. So if $x = y$ is a formula of two free variables, $\forall y(x = y)$ is a formula of just one free variable. The variable $y$ inside is bound.

We can also combine propositional connectives and quantifiers in any order.

$$\forall x(x = a) \rightarrow \forall x(f(x, a) = f(a, a))$$

$$\forall x \bot \land \forall x(x = f(a, a) \rightarrow \forall y(f(y, a) = f(x, a) \land y = y))$$  \hspace{1cm} (80)

Here’s a more formal statement of all these formation rules.

1. **Def.** The set of formulas of the first-order language with signature $L$ and at most $n$ free variables, which we label $L_n$, is defined recursively as follows:
(a) Each identity formula with terms from Term_{L_0}^n \text{ is a formula in } L_n.

(b) Let \( P \in \text{Prop}_k \) be a \( k \)-place propositional connective, and let \( A \in (L_n)^k \) be a sequence of \( k \) formulas. Then \( P[A] \) is a formula in \( L_n \). (That is to say, if \( A_1, \ldots, A_k \) are each formulas in \( L_n \), then \( P[A_1, \ldots, A_k] \) is a formula in \( L_n \).)

(In other words, any propositional sentence in \( \text{Prop}_n \) is in \( L_n \).)

(c) For any formula \( A \) in \( L(n+1) \), the universal generalization \( \forall x_{n+1} A \) is in \( L_n \).

(d) Nothing is in \( L_n \) unless it is formed by finitely many applications of steps (a)–(c).

2. **Def.** \( L_0 \) is the set of **closed formulas**, or **sentences**. \( L_1 \) is the set of **formulas of one variable**.

3. **R.** A few notes about this definition.

Since writing this definition totally officially (in terms of injective and inductive properties) is a little complicated, let’s not bother. The point is: we can prove things about first-order formulas by induction, and we can recursively define functions that take arbitrary first-order formulas to values.

Note that the first clause allows variables to occur in identities, not just closed terms.

The second clause of the definition amounts to saying that we can form conjunctions and conditionals out of first-order formulas, and that \( \bot \) is a first-order formula. Putting the clause in this more general way lets us reduce our number of clauses from three to one, but more importantly it also lets us use directly facts we already know about substitution for propositional logic to significantly simplify our definitions and proofs.

In the special case where \( P \) is a propositional variable \( p_i \), note that the formula \( P[A] \) is just the same as the formula \( A_i \). More generally, for any propositional connectives \( P \) and \( Q_1, \ldots, Q_k \), the formula \( P[Q][A] \) is the same as the formula \( P(Q_1[A], \ldots, Q_k[A]) \).

Note that we’re being a little bit more restrictive about our quantifiers than you might be used to. Officially, this definition requires quantifiers to bind variables in order, from the outside in. So, if \( x \) is the first variable and \( y \) the second, then technically this counts as a formula in \( L_2 \)

\[ \forall x \forall y \ x = y \]

But, technically, this doesn’t count as a formula:
5.1 Syntax

∀y∀x x = y

In practice, we won’t really worry about this much. The convention doesn’t really make any difference, because you can always just reorder your variables. But restricting our variables like this makes our proofs more convenient.

4. Example. These are examples of first-order formulas in the language of arithmetic with no free variables.

0 = suc 1
⊥
∀x(x + 0 = x)

0 = 0 → ∀x x = 0
∀x∀y((x + y = y + x) ∧ ∀z(x + y = z → suc x + suc y = suc z))

5. Def. (Some useful abbreviations.)

(a) For any formula A, let ¬A be A → ⊥.
(b) For any terms a and b, let a ≠ b be ¬(a = b).
(c) For any formula A ∈ L(n + 1), let ∃xA be ¬∀x¬A.

6. Def. Let A ∈ Ln, and let t ∈ (Term_L m)n be a sequence of terms. Then the substitution instance A[t] is defined recursively:

(a) For an identity a = b, we appeal to the definition of substitution for identities (Def. 4.3.8):

(a = b)[t] is (a[t] = b[t])

(b) For a k-place propositional connective P, and a sequence of formulas A ∈ (Ln)k, we appeal to the definition of substitution for propositional logic (Def. 3.2.2):

(P[B])[t] = P[B_1[t], ..., B_k[t]]

(c) For quantificational formulas, we have something new:

∀x_{m+1}A = ∀x_{m+1}A[t_1, ..., t_n, x_{m+1}]
Note that this means the *bound* variable $x_{n+1}$ is left alone by substitutions. Only the *free* variables which were “visible from the outside” of the formula get modified. But also notice that because of our conventions for ordering variables, this has the upshot that bound variables get automatically relabeled when we make substitutions that change the number of free variables in the formula. We always want to bind the last variable, and the number of variables in the formula can change from $n + 1$ to $m + 1$ when we make substitutions, so which variable is last switches from $x_{n+1}$ to $x_{m+1}$.

7. Example. Suppose $f$ is a two-place operation, is a one-place operation, and $a$ is a name. Let $A$ be this 2-variable formula:

$$x = y \rightarrow \forall z(f(x, y) = f(x, z) \rightarrow x = z) \quad (85)$$

In that case $A[f(x, x), ga]$ is the one-variable formula

$$f(x, x) = ga \rightarrow \forall y(f(f(x, x), ya) = f(f(x, x), y) \rightarrow f(x, x) = y) \quad (86)$$

(Check this by applying the definition step by step.)

8. Write out the following substitution instance explicitly.

$$(\forall z(f(x, y) \rightarrow f(y, z) \rightarrow f(x, z)))[gy, f(x, a)] \quad (87)$$

9. * For any formula $A \in Lk$, any sequence of terms $a$ in $(\text{Term } n)^k$ and any sequence of terms $b$ in $(\text{Term } m)^n$,

$$A[a_1[b], \ldots, a_k[b]] = A[a][b] \quad (88)$$

10. For any formula $A \in Ln$, there is a finite signature $L_0$ such that $L$ extends $L_0$, and $A$ is (syntactically identical to) a formula in $L_0^n$.

### 5.2 Interpretations

Next we’ll define how to interpret a first-order formula in a structure. This will extend our definition of the interpretations for terms, and it will also use our definition of interpretations for propositional formulas.
The basic idea is that the interpretation of a sentence in a structure is a truth-value, and more generally, the interpretation of a formula with \( n \) variables is an \( n \)-ary relation on a domain. An \( n \)-ary relation on \( D \) is a function from length-\( n \) sequences in \( D \) to truth-values. Since an assignment is a sequence of elements of \( D \), another way of putting this is that we interpret a formula in a structure by choosing its truth-value for each assignment.

1. **Def.** Let \( \text{Rel}_D n = D^n \rightarrow \mathbb{2} \), the set of \( n \)-place relations on \( D \).
   
   (Note that \( \text{Rel}_D 1 \) corresponds to the set of characteristic functions on \( D \), or in other words it’s in one-to-one correspondence with the power set of \( D \). A zero-place relation, an element of \( \text{Rel}_D 0 \), corresponds to an element of \( \mathbb{2} \), that is, a truth-value.)

2. **Def.** Let \( S \) be an \( L \)-structure. The interpretation for first-order formulas gives us (for each \( n \in \mathbb{N} \)) a function

\[
\llbracket \cdot \rrbracket_S : L^n \rightarrow \text{Rel}_{D_S} n
\]

That is, if \( A \in L^n \) and \( d \in D_S^n \) is an assignment, then \( \llbracket A \rrbracket_S^d \) is the truth-value of \( A \) for \( d \) in \( S \). This function is defined recursively as follows. There are three cases.

(a) For identity formulas,

\[
\llbracket a = b \rrbracket_S^d = \top \iff \llbracket a \rrbracket_S^d = \llbracket b \rrbracket_S^d
\]

Here we’re using the definition of \( \llbracket a \rrbracket_S^d \) and \( \llbracket b \rrbracket_S^d \) for terms from Part 4. (This is the same definition as we used earlier for the truth of an identity formula in a structure.)

(b) For propositional formulas,

\[
\llbracket P[B_1, \ldots, B_n] \rrbracket_S^d = \llbracket P \rrbracket (\llbracket B_1 \rrbracket_S^d, \ldots, \llbracket B_n \rrbracket_S^d)
\]

Here \( \llbracket P \rrbracket \) is the truth function denoted by \( P \), defined in Part 3. (So in other words, we calculate the truth-value of a formula that results from plugging some formulas \( B_1, \ldots, B_n \) into a propositional formula \( P \) by plugging the truth-values of \( B_1, \ldots, B_n \) into the truth-function denoted by \( P \).)
(c) For universal generalizations,

\[ \llbracket \forall_{\tau+1} A \rrbracket^d_S = T \iff \llbracket A \rrbracket^{(d_1, \ldots, d_n, d^*)}_S = T \text{ for every } d^* \in D_S \]  \hspace{1cm} \text{(92)}

(So for example, a universally quantified formula \( \forall x A[x] \) is true in a structure \( S \) iff \( A[x] \) is true for each possible assignment of a value to \( x \) in the domain of \( S \).)

3. **Def.** A sentence \( A \) is **true** in \( S \) iff \( \llbracket A \rrbracket^0_S = T \). Otherwise \( A \) is **false** in \( S \).

4. **Example.** We'll show from the definitions that \( \forall x (x = a) \rightarrow a = b \) is true in every structure \( S \).

First, by the clause for propositional formulas, we know

\[ \llbracket \forall x (x = a) \rightarrow a = b \rrbracket^0_S = \llbracket \forall x (x = a) \rrbracket^0_S \rightarrow \llbracket a = b \rrbracket^0_S \]  \hspace{1cm} \text{(93)}

(On the right hand side, the symbol \( \rightarrow \) is the conditional truth-function. We get this equation, because the sentence on the left comes from substituting the components \( \forall x (x = a) \) and \( b = a \) into the propositional formula \( p \rightarrow q \), and the truth-function denoted by \( p \rightarrow q \) is just the truth-function \( (\rightarrow) \).)

Now, suppose this value is \( \bot \). That implies:

\[ \llbracket \forall x (x = a) \rrbracket^0_S = T \quad \text{and} \quad \llbracket a = b \rrbracket^0_S = \bot \]  \hspace{1cm} \text{(94)}

The right side tells us that \( \llbracket a \rrbracket^0_S \) and \( \llbracket b \rrbracket^0_S \) are distinct. That is, \( a_S \) and \( b_S \) are different elements of the domain \( D_S \). The left side tells us that, for any element of the domain \( d \in D_S \),

\[ \llbracket x = d \rrbracket^{(d)}_S = T \]  \hspace{1cm} \text{(95)}

In particular, consider \( d = b_S \). Then we would have to have \( \llbracket x \rrbracket^{(b_S)}_S \) and \( \llbracket a \rrbracket^{(b_S)}_S \) be the same thing. We also know (by the interpretation function for terms)

\[ \llbracket x \rrbracket^{(b_S)}_S = b_S \quad \text{and} \quad \llbracket a \rrbracket^{(b_S)}_S = a_S \]  \hspace{1cm} \text{(96)}

But we already showed that \( a_S \) and \( b_S \) had to be different things. So this is a contradiction, and we can conclude that the original conditional sentence must be true in every structure.
5. For each of the following sentences, use the definitions to check whether it is true in all structures.

(a) $\forall x \exists y (fx = y)$

(b) $\forall x (fx = a)$

(c) $\forall x \forall y \forall z ((x = y \land y = z) \rightarrow fx = fz)$

6. Def.

(a) Let $X$ be a set of sentences and $A$ be a sentence. Then $A$ validly follows from $X$ (for short, $X \models A$) iff there is no structure $S$ such that each sentence in $X$ is true in $S$, and $A$ is false in $S$. (We also sometimes write $X \models_L A$ to be explicit about the language of $X$ and $A$.)

(b) A structure $S$ is a model of a set of sentences $X$ iff $\llbracket A \rrbracket_S = T$ for every sentence $A \in X$.

(c) A set of sentences $X$ is jointly satisfiable (or semantically consistent) iff $X$ has a model.

7. $\forall x f(x, a) \models \exists x f(x, a)$

8. For any $L$-structure $S$, and any formula $A[x] \in L_1$, the sentence $\exists x A[x]$ is true in $S$ iff there is some $d \in D_S$ such that $\llbracket A[x] \rrbracket_S(d) = T$.

9. For any $L$-structure $S$, and any formula $A[x] \in L_1$, $\forall x A[x]$ is false in $S$ iff $\exists x \neg A[x]$ is true in $S$.

10. Let $t$ be a name, and $f$ a two-place operation. The following sentences are jointly satisfiable.

$$\forall x f(x, x) \neq t \quad \forall x \exists y f(x, y) = t \\
\forall x \forall y \forall z (f(x, y) = t \rightarrow (f(y, z) = t \rightarrow f(x, z) = 0)) \tag{97}$$

11. Let $L^+$ be a signature that extends $L$. Each formula in $L_n$ can be naturally identified with a formula in $L^+_n$. Let $S^+$ be an $L^+$ structure. Let $S$ be the reduction of $S^+$ to $L$ (in the sense of Exercise 4.2.14). Then for any formula $A$ in $L_n$, $\llbracket A \rrbracket_S = \llbracket A \rrbracket_{S^+}$.

12. * For any formula $A$ in $L(n+1)$, any closed $L$-term $a$, and any $d_1, \ldots, d_n \in D_S$,

$$\llbracket A[x_1, \ldots, x_{n+1}] \rrbracket_S(\llbracket a \rrbracket_S, d_1, \ldots, d_n) = \llbracket A[a, x_1, \ldots, x_n] \rrbracket_S(d_1, \ldots, d_n) \tag{98}$$
In particular, if $A[x]$ is a formula of one variable, then

$$\llbracket A(x) \rrbracket_S (\llbracket a \rrbracket_S) = \llbracket A(a) \rrbracket_S$$

(Recall 4.2.13.)

13. Let $X$ be a set of $L$-sentences, let $A[x]$ be a formula in $L_1$, and let $a$ be a closed $L$-term. In that case:

$$\text{if } X \models \forall x A[x] \text{ then } X \not\models A[a]$$

(100)

14. Suppose that $X$ is a set of $L$-sentences, and $A[x]$ is a formula in $L_1$. Let $L + a$ be the signature that extends $L$ by adding one name $a$ that is not in the original signature $L$. In that case:

$$\text{if } X \models_{L + a} A[a] \text{ then } X \models_{L} \forall x A[x]$$

(101)

Hint. Suppose $\forall x A[x]$ is false in an $L$-structure $S$. Then there is an $(L + a)$-structure $S^+$ such that $A[a]$ is false in $S^+$, and $S$ is the reduction of $S^+$.

15. Def. Let $S$ be an $L$-structure. The **theory of** $S$, written $\text{Th } S$, is the set of $L$-sentences which are true in $S$.

16. Suppose $h : S \rightarrow T$ is an isomorphism. For every formula $A \in L_n$,

$$\llbracket A \rrbracket_S^d = \llbracket A \rrbracket_{T}^{hsd}$$

(102)

17. If $S \cong T$, then $\text{Th } S = \text{Th } T$.

### 5.3 Proofs

To prove things in first-order logic, we’ll still have all of our old proof rules from propositional logic and from the logic of identity. We also have two new ones: an introduction rule and an elimination rule for the universal quantifier. The intro rule—Universal Generalization—uses a new idea. Think about how we’ve proved universal generalizations in the past. To prove “Every function is Blah”, we say something like this: “Let $f$ be a function. Then (reasoning here). Therefore $f$ is Blah.” The key idea here is that we introduce $f$ as an **arbitrary name**. It doesn’t pick out any function in particular, but rather it’s a placeholder in our reasoning for
any function. If we can prove that an arbitrary function has a certain property, then we can conclude that every function has that property.

The main tricky thing about this is that when we consider what’s provable in one language, we can appeal to proofs in another language, which adds in an extra name. The rule of universal generalization says: if $X$ is a set of $L$-sentences, and $A[a]$ is an $L$-formula of one variable, and $X \vdash_{L+a} A[a]$, then $X \vdash_{L} \forall x A[x]$. Note the subscripts, which indicate which language we’re proving things in. We won’t always write these in explicitly, but for Universal Generalization steps it’s a good idea. In particular, $L + a$ here means a language which adds in a new name $a$, which was not already part of $L$’s signature. Note that since $X$ and $A[x]$ are in the language $L$, it follows that they don’t contain any occurrences of $a$. (This is the sense in which $a$ is “arbitrary”: we don’t assume anything about it in any of our premises.) But of course $A[a]$ may contain occurrences of $a$.

The language-subscript on the $\vdash$ symbol now works like a bookkeeping device for keeping track of which arbitrary names we’re using in our proof. This is very similar to our use of sequents for keeping track of which assumptions our proof is using—indeed, introducing an arbitrary name into a proof is conceptually very similar to introducing an assumption. In both cases we eventually want to “discharge” the names and assumptions from our proof. We’ll want to make sure that the last line of our proof really uses just the language of our original premises and conclusion.

In this section we’ll introduce the proof system and show that it’s sound, and then prove a few important “finiteness properties” of proofs. The last exercises in this section are preparation for the completeness theorem.

1. **Def.** An $L$-sequent is a pair of a set of $L$-sentences $X$ and an $L$-sentence $A$, which we write $X \vdash_{L} A$.

2. **Def.** A proof is a finite sequence of $L$-sequents for various signatures $L$, where each sequent $X \vdash_{L} A$ is one of the following kinds of step.

   (a) **Propositional Logic.** $X \vdash A$ is provable in propositional logic. (In this case, we are using the fact that $L$-sentences are also sentences of propositional logic, with arbitrary sentences of $L0$ considered as “sentence letters”.)

   (b) **Identity Logic.** $Y \vdash A$ is provable in the logic of identity, for some set of identities $Y \subseteq X$. 


(c) **Quantifier Elimination** (Universal Instantiation). \( A \) is \( B[a] \) for some \( L1 \) formula \( B \) and some \( L \)-term \( a \), and \( X \vdash \forall x B \) occurs as an earlier step.

(d) **Quantifier Introduction** (Universal Generalization). \( A \) is \( \forall x B \) for some \( L1 \) formula \( B \); the signature \( L + a \) extends \( L \) by adding a single name \( a \) not in the signature \( L \); and \( X \vdash B[a] \) occurs as an earlier step.

(e) **Cut.** \( X \) is \( Y \cup Z \) for some sets of sentences \( Y \) and \( Z \), and \( Y \vdash B \) and \( Z \vdash A \) occur as earlier steps.

(The Cut rule is added explicitly in this system, rather than being derived from more basic rules, because this helps us to “glue together” the things we prove in the “smaller” logics of propositional formulas and identities.)

In shorter form:

\[
\begin{align*}
X \vdash A & \quad \text{Prop} \quad \text{if } X \vdash A \text{ in propositional logic} \\
X, Y \vdash A & \quad \text{Id} \quad \text{if } Y \vdash A \text{ in identity logic} \\
X \vdash B[a] & \quad X \vdash \forall x A & \quad \forall \text{Intro} \quad \text{where } a \text{ is a name not in } L \\
X \vdash \forall x A & \quad \forall \text{Elim} \quad \text{for any term } a \\
X \vdash A & \quad \forall \lambda \alpha \quad \forall \text{Elim} \quad \text{for any term } \lambda \alpha
\end{align*}
\]

(103)

3. **Example.** Let \( L \) be a language with a single name \( a \). We’ll prove: \( \forall x(x = a) \vdash \forall x \forall y(x = y) \).
5.3 Proofs

∀x(x = a) ⊢ ∀x(x = a)  Ass

∀x(x = a) ⊢ b = a  ∀Elim

b = a, c = a ⊢ b = c  Id

∀x(x = a), c = a ⊢ b = c  Cut

∀x(x = a) ⊢ c = a  ∀Elim

∀x(x = a) ⊢ b = c  Cut

∀x(x = a) ⊢ ∀x(b = x)  ∀Intro

∀x(x = a) ⊢ ∀x∀y(x = y)  ∀Intro

4. Def.

(a) A sequent \( X \vdash A \) is **provable** iff it is the last line of some proof. (As usual, we also say \( X \vdash A \) to say that the sequent is provable.)

(b) A set of sentences is **(L-)inconsistent** iff \( X \vdash \bot \). Otherwise \( X \) is **(L-)consistent**.

(c) When we need to distinguish between our three different proof systems, we’ll use qualified expressions like “Prop-consistent”, and “Id-provable”.

5. Let \( X \) be a set of sentences. If \( X \) is consistent, then it is Prop-consistent, and if \( Y \subseteq X \) is a set of identities, then \( Y \) is Id-consistent.

6. (Soundness.) If \( X \vdash A \) then \( X \vDash A \).

   (Use induction on proof length.)

7. For any structure \( S \), its theory \( ThS \) is consistent.

8. * If \( X \vdash A \) and \( L^+ \) extends \( L \), then \( X \vdash A \).

   (Use induction on proof length.)

9. * If \( X \vdash A \), then there is a finite subset \( X_0 \subseteq X \) and a finite signature \( L_0 \) such that \( L \) extends \( L_0 \), and \( X_0 \vdash A \).

   (Use induction on proof length.)
10. Let $L$ be a signature and $X$ a set of $L$-sentences. Suppose that for every finite signature $L'$ that $L$ extends, and for every finite subset $X^- \subseteq X$ consisting of $L'$-sentences, $X^-$ is $L'$-consistent. Then $X$ is $L$-consistent.

11. Suppose that we have a sequence of signatures $L_0 \leq L_1 \leq L_2 \leq \ldots$, and we also have a sequence of sets of sentences $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots$, such that for each $n \in \mathbb{N}$, $X_n$ is a set of $L_n$-sentences, and each $X_n$ is $L_n$-consistent. Let $L^+$ be the signature that contains just the symbols that occur in any $L_n$ (with the same arities) and let $X^+ = \bigcup_n X_n$ be the set of all sentences that occur in any $X_n$. Then it follows that $X^+$ is $L^+$-consistent.

12. Suppose $L$ contains at least one name. Let $X$ be a set of $L$-sentences. Let $a$ be a name that does not occur in $L$.

(a) Let $A$ be an $L$-sentence. We can also consider $A$ as an $L_1$ formula in which the variable $x$ simply never occurs. If $X \vdash_{L} \forall x A$, then $X \vdash_{L} A$.

Hint. If a variable doesn’t occur at all in $A$, then substituting any term at all into $A$ doesn’t make any difference.

(b) Let $A$ be an $L$-sentence. If $X \vdash_{L+a} A$, then $X \vdash_{L} A$.

(c) Suppose that $L^+$ extends $L$ by adding just finitely many names. Suppose that $X$ is an $L$-consistent set of sentences in $L$. Then $X$ is also $L^+$-consistent.

(d) Let $A$ be a formula in $L_1$. Suppose

\[ X, \neg A[a] \vdash_{L+a} \bot \quad \text{and} \quad X, \forall x A \vdash_{L+a} \bot \quad \text{ (105)} \]

Then $X \vdash_{L} \bot$.

(e) If $X$ is $L$-consistent, then at least one of $X \cup \{ \neg A[a] \}$ or $X \cup \{ \forall x A \}$ is $(L+a)$-consistent.

5.4 Completeness

The completeness proof for predicate logic uses the same basic idea as the others we’ve done: we start with a consistent theory, and we use that as our raw materials to construct a model for those sentences. As in the completeness proof for identity, we use equivalence classes of terms as the domain. Now that we have quantifiers, though, there is something else we need to be careful about: we need to make sure that every generalization has a witness, some element of the domain which provides an
instance for that generalization. If our theory implies \( \exists x \, A \), there had better be some element of the domain that satisfies \( A \), which means, since our domain is built out of terms, there had better be some term \( a \) such that \( A[a] \) is true in our structure. So before we construct our structure by taking equivalence classes, first we need to make sure we have enough terms to go around: we need to come up with a name for something that witnesses each generalization. The way we’ll do this is by, first, adding a bunch of extra names to our original language, and second, adding a bunch of extra sentences to our original theory: for each formula \( A \in L_1 \), we’ll add in something that implies that if the theory implies \( \exists x \, A \), then it also implies \( A[a] \) for a specific term \( a \). Or, dually (since we’re using universal quantifiers as basic, rather than existential quantifiers), we want to ensure that for each formula \( A[x] \), we have some name \( a \) such that \( A[a] \rightarrow \forall x \, A[x] \) is guaranteed to be true. We call a theory like this “witness-complete”.

So, starting with a consistent set of sentences \( X \), in a countable language \( L \), the construction of a model for \( X \) goes in three stages.

1. First we extend \( X \) to make a witness-complete, consistent set of sentences, by adding in names one by one.
2. Next we use the ideas from our completeness proof for propositional logic to extend this to a negation-complete, witness-complete, consistent set.
3. Finally, we apply the completeness proof for identity logic to get a structure from the identities that appear in this set.

Once we’ve done all this, we just need to check that this recipe really does verify all the sentences in the set we built up, and thus all the sentences in \( X \).

1. **R.** Throughout this section, suppose \( L \) is a countable signature with at least one name.

2. **Def.**
   
   (a) A set of \( L \)-sentences \( X \) is **negation-complete** iff for every sentence \( A \in L_0 \), either \( A \in X \) or \( \neg A \in X \).
   
   (b) A set of \( L \)-sentences \( X \) is **witness-complete** iff for every formula \( A \in L_1 \), either \( \neg A[a] \in X \) for some term \( a \) in \( L \), or \( \forall x \, A[x] \in X \).
   
   (c) Call a set of sentences **nice** iff it is consistent, negation-complete, and witness-complete.
3. Let $L$ be a countable signature including at least one name, and let $X$ be a consistent set of $L$-sentences. Our goal in this problem is to show how to extend the language by adding new names, and extend $X$ by adding new sentences in the extended language, in order to get a witness-complete, consistent set. Our strategy for this is to add in a witness for each formula in the language, one by one.

Let $L^+$ be the signature that results from extending $L$ with a countably infinite set of new names. Since $L^+$ is also a countable signature, we know that there are countably many formulas of one variable in $L^+$. So we can enumerate all the formulas: for each $n \in \mathbb{N}$, let $A_n[x]$ be the $n$th formula in $L^+$.

Suppose $Y$ is a set of sentences in $L^+$, and $n$ is a natural number. Say $Y$ is **witness-complete up to $n$** iff for every $i < n$, either $\neg A_i[a] \in Y$ for some term $a$, or else $\forall x A_i[x] \in Y$. (In other words, each of the first $n$ formulas have a witness.)

(a) For each $n \in \mathbb{N}$, there is some signature $L_n$ that extends $L$ by adding finitely many names from $L^+$, and some set $Y_n$ of $L_n$-sentences that includes $X$, such that $Y_n$ is $L_n$-consistent and witness-complete up to $n$.

*Hint.* Use induction on $n$, and use 5.3.12.

(b) Let $X^+ = \bigcup_n Y_n$ be the set of all sentences which occur in any $Y_n$. Then $X^+$ is an $L^+$-consistent, witness-complete extension of $X$.

*Hint.* Use 5.3.11.

4. For any witness-complete consistent set of $L$-sentences $X$, there is a nice set $Y \supseteq X$.

5. **R.** This much shows that, for any consistent set of $L$-sentences $X$, there is a nice set of $L^+$-sentences that includes $X$. Now we’ll show that we can construct a model of any nice set.

Here’s a reminder of some useful facts from earlier sections.

(a) (3.7.11, slightly reworded) Suppose $X$ is negation-complete and Prop-consistent, and $P$ is any $n$-ary propositional formula. Say $\text{char}_X A = \top$ if $A \in X$ and $\bot$ otherwise. Then:

$$\llbracket P \rrbracket (\text{char}_X A_1, \ldots, \text{char}_X A_n) = \text{char}_X P[A]$$  \hspace{1cm} (106)

(Here $\llbracket P \rrbracket$ is the truth-function that $P$ denotes, as defined in Part 3.)
5.4 Completeness

(b) (4.3.14) Suppose $X$ is any set of identities. Then there is a structure $S$ such that, for any identity sentence $A$, $⟦A⟧_S = ⊤$ iff $X ⊢ A$ (in identity logic). Furthermore, every element of $D_S$ is equal to $\bar{a}$ for some term $a$; and if $a$ is any $n$-variable term and $b$ is any sequence of $n$ terms, then

$$⟦a⟧_S(\bar{b}_1, \ldots, \bar{b}_n) = ⟦a[b]⟧_S$$  \hspace{1cm} (107)

(c) (5.2.6) If $L^+$ extends $L$, then any $L^+$ structure $S^+$ has a reduction $S$ such that the very same $L$-sentences are true in both $S^+$ and $S$.

6. Suppose that $X$ is nice.

(a) For any sentence $A$, $A \in X$ iff $X ⊢ A$.

(b) For any one-variable formula $A[x]$,

$$\forall x.A[x] \in X \iff A[a] \in X \text{ for every term } a$$  \hspace{1cm} (108)

7. Suppose that $X$ is nice. Let $\text{char}_X.A = ⊤$ iff $A \in X$. Then there is a structure $S$ such that every element of the domain is $\bar{a}$ for some term $a$, and furthermore:

(a) For every identity sentence $A$,

$$⟦A⟧_S = \text{char}_X.A$$  \hspace{1cm} (109)

(b) For every $n$-variable identity formula $A$, and any sequence of $n$ terms $\bar{a}$,

$$⟦A⟧_S(\bar{a}_1, \ldots, \bar{a}_n) = \text{char}_X.A[\bar{a}]$$  \hspace{1cm} (110)

(c) For every formula $A$ in $L_n$, and any length-$n$ sequence of terms $\bar{a}$,

$$⟦A⟧_S(\bar{a}_1, \ldots, \bar{a}_n) = \text{char}_X.A[\bar{a}]$$  \hspace{1cm} (111)

(d) For every sentence $A$, $A$ is true in $S$ iff $A \in X$.

8. (Henkin’s Lemma.) If $X$ is any consistent set of sentences, then $X$ has a model.

9. (Completeness.) If $X ⊬ A$ then $X ⊬ A$. 
5.5 Consequences of Completeness

1. (Downward Löwenheim-Skolem Theorem.) Suppose that \( X \) is a set of sentences in a language with a countable signature, and \( X \) has a model. Then \( X \) has a model with a countable domain.

   *Hint.* Look at the structure we used to prove Completeness.

2. **Def.** A theory \( X \) is **categorical** iff all models of \( X \) are isomorphic. That is, if \( S \) and \( T \) are structures in which each sentence in \( X \) is true, then \( S \cong T \).

3. If \( X \) has an uncountably infinite model, then \( X \) is not categorical.

4. (Compactness.) Suppose that \( X \models A \). Then \( X \) has some countable subset \( X_0 \) such that \( X_0 \models A \).

5. If every finite subset of \( X \) has a model, then \( X \) has a model.

6. (Non-Standard Models of Arithmetic.)

   (a) Let \( S \) be a structure for the language of arithmetic, and suppose that for some element \( d \) of the domain of \( S \), \( d \neq \llbracket n \rrbracket_S \) for any numeral \( n \). Then \( S \) is not isomorphic to \( \mathbb{N} \).

   (b) Extend the language of arithmetic by adding a new name \( a \). We know that Th \( \mathbb{N} \) is still consistent in the extended language. Consider the set of sentences:

   \[
   X = \text{Th} \mathbb{N} \cup \{ a \neq 0, a \neq 1, a \neq 2, \ldots \} \tag{112}
   \]

   That is, we add to Th \( \mathbb{N} \) the sentence \( \neg(a = n) \) for each numeral \( n \). Show that \( X \) is consistent.

   (c) Th \( \mathbb{N} \) has a model which is not isomorphic to \( \mathbb{N} \). That is, Th \( \mathbb{N} \) is not categorical.

7. (Undefinability of Finiteness)

   (a) For any \( n \in \mathbb{N} \), there is a sentence \( A \) such that, for every structure \( S \), \( A \) is true in \( S \) iff \( D_S \) has at least \( n \) elements.

   (b) Suppose that \( X \) is a theory that has models of every finite size. Then \( X \) has an infinite model.
6 The Inexpressible

6.1 Notes on Use and Mention

In this section, we’ll prove several important results about the limits of what can be said: Tarski’s theorems about the undefinability of satisfaction and truth, and a simple version of Gödel’s theorem that no reasonably strong theory can be complete and consistent.

The main new idea (and the main conceptual difficulty) is that we’re going to be looking at languages that are intuitively about language. So far, we’ve been discussing notions like truth and satisfaction (i.e., truth-at-an-assignment) and provability entirely in an informal “meta-language”: English, embellished with a few formal notations. (This meta-language is “informal” in the sense that we haven’t given a precise definition of its syntax and semantics, the way we did for, say, the language of propositional logic.) Now we’re going to look at what happens when we try to introduce these ideas about syntax and semantics and proof into the formal “object language” (so-called because it is our object of study). That is, we’re going to consider formal theories which are stated in languages like predicate logic, and which have formulas like $\text{true}(s)$ that intuitively say that a certain sentence $s$ is true, or $\text{sat}(x,f)$ which intuitively says that a certain thing $x$ satisfies an open formula $f$.

To do this right, we need to be a bit more careful about the distinction between use and mention—between what a bit of language says and what it is. Our standard equipment for this is quotation marks. Here’s a simple example.

- Obama is a president.
- “Obama” is a name of a president.

The quotation marks indicate that we are mentioning the bit of language spelled O-b-a-m-a, a certain three-syllable word, rather than using that bit of language to discuss a certain person.

For any phrase of English, we can write out that phrase, and put quotation marks around it, in order to come up with a name for that phrase. “Snow is white” is a sentence, ““Snow is white”” (note the double-quotations marks) is a name for the sentence “Snow is white”. And ““““Snow is white””” is a name for that name, ““Snow is white””.
Take this famous biconditional (which is an instance of what’s called “the T-schema” or “Convention T”):

- “Snow is white” is a true sentence iff snow is white.

On the left hand side we are mentioning a certain sentence, by using a quotation-name to refer to it; on the right hand side we are using that very sentence to say something about snow, rather than saying something about a sentence. Here’s another version:

- Fido satisfies the open formula “x is a dog” iff Fido is a dog.

In this case we’re using “Fido” as a name for an individual, and ““x is a dog”” as a name for a certain open formula, namely the formula “x is a dog”. And we are using (not mentioning) the sentence “Fido is a dog” to say that Fido is a dog. Hopefully that’s not completely obscure—but you’ll have more opportunities to get the hang of things as we go.

We’ll be looking at formal languages (mainly predicate logic) which have a similar kind of resource to English quotation marks: for each formula of the language, there is a term in the language which is a quotation of that formula (or a “code”). Or to put it the other way around, there is a certain stock of quotation terms $Q$, each of which denotes a certain formula. This denotation function will be onto, in the sense that every formula has some quotation term that denotes it. We’re not going to worry (for now) about whether quotation terms are simple names or complex terms built out of function symbols. The important thing is just that somehow or other we can label every formula.

(Later on we’ll look at how even languages with pretty simple resources for describing numbers can have quotations for their own terms, using the idea of Gödel numbering. The basic idea behind that is one we’ve already looked at: we can put the formulas in a countable language in one-to-one correspondence with numbers, in a natural way.)

So now let’s look at how to formalize the T-schema. Say we have an open formula $true[x]$ of one variable. Then whenever $q$ is a quotation term that denotes a certain sentence $A$, this ought to be a true sentence:

$$true[q] \leftrightarrow A$$  (113)
(Notice that some other things we might try to say don’t make grammatical sense, like \( \text{true}[A] \leftrightarrow A \). We can’t say this, because \( A \) is a sentence, rather than a term.) So we’ll be looking at theories that have every instance of the T-schema as a logical consequence.

There is another important version of the T-schema which uses satisfaction rather than truth. Say we have an open formula \( \text{sat}[x,y] \) of two variables. Then whenever \( q \) is a quotation-term for an open formula \( A[x] \), and \( a \) is any term, this ought to be a true sentence:

\[
\text{sat}[a,q] \leftrightarrow A[a] \tag{114}
\]

That is, if \( a \) satisfies \( q \), then plugging \( a \) into the formula \( A[x] \) should yield a true sentence, and vice versa. Let’s call this the Sat-schema. (The Sat-schema formalizes this result we proved earlier: \( \llbracket A[x] \rrbracket (\llbracket a \rrbracket) = \llbracket A[a] \rrbracket \).

In fact, what we’ll show is (roughly) that any theory that implies every instance of either the T-schema or the Sat-schema is logically inconsistent. The reason for this has to do with the Liar Paradox. What we can show is that, in fact, if the theory has enough resources to describe its own syntax (in a sense we’ll make more precise), then it follows that it contains “Liar sentences”: sentences which are equivalent to their own untruth. But this is getting a little ahead of ourselves.

We’ll be working toward the following three theorems:

- Any theory that includes a formula that obeys the Sat-schema is inconsistent.
- Tarski’s Theorem. Any theory that includes a formula that obeys the T-schema, and which also has a way of defining substitution, is inconsistent.
- Gödel’s Theorem. Any consistent theory that includes a formula for provability includes sentences that can be neither proved nor refuted.

In short, Tarski’s Theorem shows that no consistent theory can describe its own syntax and semantics. And Gödel’s theorem shows that no proof system which is simple enough to be described within a theory is also powerful enough to prove every truth.

(Later on—if we have time—we’ll see that in fact the existence of quotation terms, defining substitution, and defining provability, are really not very demanding re-
requirements: any theory that can prove a few basic things about numbers will do the job.

6.2 Basic Facts About Equivalence

1. **R.** Let’s fix some signature \( L \). “Formula” means \( L \)-formula, “sentence” means \( L \)-sentence, and “term” means closed \( L \)-term.

2. **Def.**

   (a) For any sentences \( A \) and \( B \), let \( A \leftrightarrow B \) be an abbreviation for \( (A \rightarrow B) \land (B \rightarrow A) \).

   (b) A **theory** is a set of sentences closed under logical consequence. That is, for any sentence \( A \), if \( X \vdash A \) then \( A \in X \).

   (c) For a theory \( X \), we say \( A \) and \( B \) are **equivalent in \( X \)** iff \( \vdash_X A \leftrightarrow B \). This is abbreviated \( A \equiv_X B \), or just \( A \equiv B \) if the choice of \( X \) is clear enough in context.

3. For any theory \( X \), and any sentences \( A \) and \( B \), \( A \equiv_X B \) iff whenever \( S \) is a model of \( X \), \( \overline{A}_S = \overline{B}_S \).

4. For any theory \( X \), the relation \( \equiv_X \) is an equivalence relation: that is, for any sentences \( A, B, C \):

   (a) \( A \equiv A \)

   (b) If \( A \equiv B \) then \( B \equiv A \).

   (c) If \( A \equiv B \) and \( B \equiv C \) then \( A \equiv C \).

5. For any theory \( X \):

   (a) \( X \vdash A \) iff \( A \equiv_X \top \) (where \( \top \) is a tautology, such as \( \bot \rightarrow \bot \))

   (b) If \( X \vdash A \) and \( X \vdash B \) then \( A \equiv_X B \).

   (c) \( X \vdash A \) iff \( A \land B \equiv_X B \) for every sentence \( B \).

   (d) \( A \equiv_X B \) iff \( \neg \overline{A}_X \equiv \neg B \)

   (e) \( X \) is inconsistent iff for all sentences \( A \) and \( B \),

\[
A \equiv_X B
\]
6.3 Satisfaction

The key idea of the satisfaction theorem is exactly parallel to the idea of Russell’s paradox (or Cantor’s Theorem). Here’s a simple and famous version. The predicate “heterological” is defined like this: a predicate is heterological iff it does not apply to itself. For example, “unpronounceable”, “French”, and “long” are heterological predicates (because they are pronounceable, English, and short, respectively). “Polysyllabic” and “beginning with ‘b’” apply to themselves, so they aren’t heterological.

Is “heterological” heterological? If it is, then it does not apply to itself. In that case it isn’t heterological. But if it isn’t heterological, then it does apply to itself. So it is heterological. This is a contradiction.

We’ll show something similar to this—not a paradox, but a theorem. We’ll show that if a theory has a predicate that obeys the Sat-schema, then it follows that it also has an open formula analogous to “heterological”—the formula “x does not satisfy x”. This implies that the theory is inconsistent.

1. Def.

(a) For any set A, we say a labeling system for A is any set T of terms with a denotation function \( h : T \rightarrow A \) which is onto.

(We could use the notation \([·]\) rather than \(h\), since effectively we’re thinking of the terms as being interpreted to stand for elements of \(A\); but I think this might make things a little more confusing.)

(b) A quotation system for sentences is a labeling system for \(L_0\), the set of all sentences. That is, it’s a set \(Q_0\) of terms we’ll call quotations, and an onto function \(h : Q_0 \rightarrow L_0\) that takes each \(q \in Q_0\) to some sentence.

(c) A quotation system for open formulas is a labeling system for \(L_1\), the set of one-variable open formulas. That its a set \(Q_1\) of quotation terms with an onto function \(h : Q_1 \rightarrow L_1\) that takes each \(q \in Q_1\) to some open formula \(A[x]\).
2. **R.** In what follows, we’ll suppose we are given some quotation system for sentences and for open formulas.

3. **Def.** A theory $X$ defines **satisfaction** iff there is some formula $\text{sat}[x,y]$ of two variables such that, for every term $a$ and any quotation term $q$ that denotes a formula $A[x],$

$$\text{sat}[a,q] \equiv A[a] \quad (117)$$

4. If $X$ defines satisfaction, then $X$ is inconsistent.

   *Hint.* Suppose that $q$ denotes the formula $\neg \text{sat}[x,x]$. Show that $\text{sat}[q,q] \equiv \neg \text{sat}[q,q].$

### 6.4 Truth

Once we have this much, what we need in order to get to Tarski’s theorem about truth is a way of getting from *truth* to *satisfaction*. We can do this if we suppose that we also have a way to describe *substitution* in the object language.

1. **Def.** A theory $X$ defines **truth** iff there is some formula $\text{true}[x]$ of one variable such that, for every quotation term $q$ that denotes a sentence $A,$

$$\text{true}[q] \equiv A \quad (118)$$

2. **Def.** A theory $X$ defines **substitution** iff there is some term $\text{sub}[x,y]$ of two variables such that, for every term $a$ and every quotation term $q$ that denotes a formula $A[x],$

$$\text{sub}[a,q] \text{ denotes } A[a] \quad (119)$$

3. If $X$ defines truth and $X$ defines substitution, then $X$ defines satisfaction.


4. (Tarski’s Theorem.) If $X$ defines truth and $X$ defines substitution, then $X$ is inconsistent.
5. **R.** In this version we supposed substitution was definable using a term. But it's worth noting that in the language of predicate logic, we don't have all that many ways to define up terms: they all have to built from putting together simple function symbols. That means that we can't use truth-functional connectives or quantifiers when we're putting together complex terms. (Richer languages that add in “lambda-abstraction” would allow us to do this kind of thing. But unfortunately we don't have time to work through the details of how that would go.) Because of this, it's worth looking at a different way in which we could define up substitution, using a *formula* instead of a term. This version is a little bit more complicated.

6. **Def.** A theory $X$ *defines substitution relationally* iff there is some formula $\text{sub}[x, y, z]$ of three variables (read this as “the substitution of $x$ into $y$ is $z$”) such that, if $a$ is a term, $q$ is a quotation term that denotes a formula $A[x]$, then there is a quotation term $q'$ that denotes the sentence $A[a]$ such that

$$X \vdash \text{sub}[a, q, q']$$

(120)

and

$$X \vdash \forall x(\text{sub}[a, q, x] \rightarrow x = q')$$

(121)

(The second part says that substituting $a$ into $q$ gives a unique result.)

7. (A useful bit of first-order logic.) If $X \vdash \forall x(A[x] \rightarrow x = a)$, then

$$\exists x.A[x] \equiv A[a]$$

(122)

8. If $X$ defines truth and $X$ defines substitution relationally, then $X$ defines satisfaction.

*Hint.* Show that if $q$ denotes the formula $A[x]$, then $\exists x(\text{sub}[a, q, x] \land \text{true}[x])$ is equivalent to $A[a]$.

9. (Tarski’s Theorem, Version 2.) If $X$ defines truth and $X$ defines substitution relationally, then $X$ is inconsistent

**6.5 The Unprovable**

1. **Def.** A theory $X$ *defines provability* iff there is some formula $\text{prov}[x]$ of one variable such that, for every quotation term $q$ that denotes a sentence $A$,
2. **Def** (repeated from earlier, as a reminder). A theory \( X \) is **complete** iff for every sentence \( A \), either \( X \vdash A \) or \( X \vdash \neg A \). Otherwise it is **incomplete**.

(This agrees with our earlier definition of negation-complete, when \( X \) is a theory.)

3. For any theory \( X \), if \( X \) defines provability and \( X \) is complete, then \( X \) defines truth.

*Hint.* If \( X \) is complete, then \( \text{prov}[x] \) in fact obeys the conditions for a truth-predicate.

4. (Gödel’s First Incompleteness Theorem.) Any theory that defines substitution and provability is either incomplete or inconsistent.

5. (Gödel’s Theorem, Alternate Version.) Any theory that defines substitution relationally and defines provability is either incomplete or inconsistent.

### 6.6 Fixed Points

In this section we’ll prove an important generalization of the core fact that we used in the previous problems: the **Diagonal Lemma**. This proves generally the existence of self-referential sentences and formulas. (It’s also the foundation for recursively defined functions in programming languages.) This is a more general version of the same “heterological” trick.

1. (Diagonal Lemma.) Suppose \( X \) defines substitution. Let \( A[x] \) be any open formula. Then there is some term \( q \) that denotes the sentence \( A[q] \).

   *Hint.* Let \( q' \) denote the formula \( A[\text{sub}[x, x]] \). Then consider the term \( q = \text{sub}[q', q'] \).

2. Suppose \( X \) defines substitution relationally. Let \( A[x] \) be any open formula. Then there is some term \( q \) that denotes a sentence \( B \) such that \( A[q] \equiv B \).

   *Hint.* Let \( q' \) denote the formula \( \exists y (\text{sub}[x, x, y] \land A[y]) \), and then use the same trick as before.

3. Explain why Tarski’s theorem follows from the Diagonal Lemma.
6.7 The Arithmetization of Syntax

What we’ve shown is that any consistent theory which defines substitution with respect to some quotation system is limited, in that it can’t prove the T-schema. Which theories are like this? The answer turns out to be surprisingly many: any theory strong enough for basic arithmetic will do.

1. There is a quotation system for the language of arithmetic.

We’ve actually already proved this: we know that the set of formulas in the language of arithmetic is countably infinite, so it’s in one-to-one correspondence with the natural numbers. We also know that the numerals are terms in the language of arithmetic, and there is one of these for each natural number. So this gives us a one-to-one correspondence between the numerals (terms in the language of arithmetic) and arbitrary formulas in arithmetic. We can think of this as fixing a numerical label for each formula—a “quotation term”, even though we don’t have anything that really looks much like quotation marks.

What isn’t so obvious is that we can do this in a reasonably nice way. We also want to fix the labels for formulas and open sentences so that the relationship between the numerical label for the formula $A[x]$ and the numerical label for the sentence $A[a]$ (where $a$ is a term) is a nice numerical relationship—one that we can define using some formula of arithmetic. This is also true—but showing it takes quite a bit more work, and we don’t have time to go into all the details.

I’ll sketch the ideas behind this:

1. $\text{ThN}$ defines substitution relationally.

(and indeed, minimal arithmetic $Q$ defines substitution relationally).

FILL THIS IN LATER.

6.8 Some Brief Notes on Decidability

Earlier we proved this version of Gödel’s First Incompleteness Theorem:
• For any theory $X$, if $X$ defines substitution and provability-in-$X$, then $X$ is inconsistent or incomplete.

But this leaves open the possibility that theories that satisfy the antecedent are unusual or perverse. Gödel also proved that they are not so special. Let’s say a theory is **decent** just in case it is both **minimally simple** in the sense that humans could ever hope to describe it, and also **minimally strong** in the sense that it includes a few basic facts of arithmetic. I’ll explain these ideas more precisely in a moment. Gödel showed that any decent theory defines substitution and provability, and thus cannot be consistent and complete. Here’s another statement of Gödel’s first theorem:

• For any theory $X$, if $X$ extends $Q$ and $X$ has a decidable set of axioms, then $X$ is inconsistent or incomplete.

I’ll explain both ideas (though I won’t give a completely precise definition of either one).

First: “$X$ extends $Q$” is a way of making precise the idea that $X$ is minimally strong. The theory $Q$—the theory of minimal arithmetic (or Robinson arithmetic)—is the name for a certain set of sentences that includes just a few simple facts of arithmetic. We won’t concern ourselves with exactly which ones. Note just that there aren’t very many of them, and they are all very obvious. A theory $X$ **extends** $Q$ iff it includes each sentence in $Q$. Or equivalently: for any sentence $A$, if $Q \vdash A$ then $X \vdash A$.

Here’s an important fact about $Q$:

• $Q$ defines substitution relationally.

There’s one other important fact about $Q$, which I’ll get to after I’ve explained “decidable”.

Now for “$X$ has a decidable set of axioms”. This is a way of making precise the idea that $X$ is minimally **simple**—the kind of theory that humans (or robots) might have a hope of precisely describing.

A set of **axioms** for a theory $X$ is some set of sentences in $X$ that has every sentence in $X$ as a consequence. That is, $Y$ is a set of axioms for $X$ iff for every sentence $A$, $X \vdash A$ if and only if $Y \vdash A$. 
The last idea that needs to be explained is a “decidable set of sentences”. This is a key idea which is central to some very important advances in 20th century philosophy. The idea is that if a set $X$ is decidable (or “effectively decidable”) then there is some precisely stateable general method you can follow, such that if you start with any sentence $A$, and you are given enough time and paper to keep track of your work, eventually following this method you will reach the result “true” if $A$ is a member of $X$, or “false” if $A$ is a member of $X$. A decidable set can be checked for membership, in some systematic way.

That description is a bit vague: what is a “precisely stateable general method”, and what does “systematic” mean? These seem like the sort of concepts that would be impossible to give a precise analysis of. But in fact, an enormously successful philosophical thesis gives just such an analysis. This is called the Church-Turing Thesis. (Or depending who you ask, “Church’s Thesis” or “Turing’s Thesis”.) Its discovery laid the basic foundation for computer science—since it essentially presented for the very first time the idea of a programming language.

I won’t give a full statement of the Church-Turing Thesis: it’s a bit complicated. The idea is that there are certain basic instructions (about five, depending on the exact formulation) which can be assembled in simple ways. The result is called a Turing machine. It’s fairly obvious that any recipe that’s formed by putting together these instructions would count as a systematic method—they’re very easy to follow. What’s far less obvious is the thesis that any systematic method can be described by putting together these basic kinds of instructions. Alan Turing boldly hypothesized that if a problem can’t be solved by a Turing machine (given enough time and space) then it can’t be solved in a general systematic way at all.

This is a bold hypothesis—and philosophers might be inclined to doubt from the start that this kind of analysis of a vague notion like “effective method” could possibly be right. (Turing’s recipe also has a bit of an ad hoc flavor: why these instructions exactly?) But it turns out that there is a lot of strong evidence that it is correct.

Alonzo Church gave a rival analysis of “effective method”, using what are called partial recursive functions. Again I won’t go into the details: suffice it to say that a partial recursive function is also made up in some simple ways from a basic inventory of four or five simple components, and the basic components of partial recursive functions are very different in spirit from the basic components in Turing machines. Church boldly hypothesized: any effectively decidable set can be decided using just partial recursive functions.
There was some debate between these two theories of computability, until a marvelous discovery was made: in fact, the notion of decidability you get from Turing machines, and the notion you get from partial recursive functions, is exactly the same. A problem is solvable using Turing machines if and only if it is solvable using partial recursive functions. This was taken to be serious evidence that they had in fact both hit on the correct answer to the question of what decidability is—since they arrived by very different paths at the exact same place.

Even more evidence has been found since then, because now that computer science has become a very practical endeavor people have come up with hundreds of different “theories of decidability”—these are programming languages like Java or Python or Haskell or Lisp, and so on. It turns out that all of these are equivalent to each other, and equivalent to Turing and Church’s original proposals, in terms of what problems they can solve. (The technical term for being equivalent in problem-solving power to Turing machines is “Turing complete”.) This is pretty wonderful.

So we’ll take the Church-Turing thesis for granted, as pretty much everyone does now. (Remember: the Church-Turing thesis isn’t a theorem—it was a philosophical hypothesis. In my view, this is probably the most successful philosophical hypothesis of the 20th century, and in the running for the most successful philosophical hypotheses in all of history.)

So to return to where we were: a “minimally simple” theory $X$ is one with a decidable set of axioms—a set of sentences $Y$ such that there is some systematic way of determining which sentences are in $Y$, such that $Y$ has just the sentences in $X$ as logical consequences.

Here’s another important fact about the theory of minimal arithmetic $Q$.

- Every decidable set of sentences is representable in $Q$.

To understand this we need to explain “representable”. The idea is that a set $X$ is representable iff there is some formula (in the language of arithmetic) that intuitively says whether or not a sentence is in $X$. Here’s how we say that more precisely.

- A formula $A[x]$ represents a set of sentences $X$ in $Q$ iff, whenever $q$ is a term that denotes some sentence $B$,

$$
\begin{align*}
&\text{if } B \in X \text{ then } Q \vdash A[q] \\
&\text{if } B \notin X \text{ then } Q \vdash \neg A[q]
\end{align*}
$$

(124)
X is representable in Q iff there is some formula A[x] that represents X in Q.

Intuitively the formula A[x] that represents X means “x is an element of X”. So whenever q denotes B, then A[q] intuitively means “B is an element of X”.

How do we justify the claim that every decidable set of sentences is representable in Q? In two steps. First, we can show that every Church-decidable set is representable in Q. This is a theorem. (To prove it we need to give a precise statement of Q and the precise definition of Church’s partial recursive functions. Once we have those, the proof is a bit long and tedious, but not terribly hard.) Second, we invoke the Church-Turing Thesis, which says that every decidable set is Church-decidable.

Now we have nearly everything we need to prove the final form of Gödel’s First Theorem. We just need to show one more fact:

- If X is consistent and complete, and X has a decidable set of axioms, then X is decidable.

Here’s a sketch of the argument for this. We’re supposing we have some program that, when given any sentence A as an input, eventually spits out “true” if A is an axiom, and “false” if A is not an axiom. We want to describe a systematic method—like a computer program—that will eventually tell us, for any sentence A, true if A is in X, and false otherwise. Call a program that does this check-axiom.

First, let’s consider what proofs are like. We know that if a sequent \( Y \vdash A \) is provable, then there’s some finite subset \( Y_0 \) of \( Y \) such that \( Y_0 \vdash A \) is provable, using a proof that only ever has finitely many assumptions on the left side of the \( \vdash \) symbol. So let’s restrict attention to those finitary proofs. A finitary proof is a finite sequence of finite sequents. Call one of these finite sequences a “proto-proof”, while a proof is a proto-proof that has the extra property that every step validly follows from earlier steps using one of our proof rules.

There is a countable infinity of proto-proofs. (Remember, if a set is countable, then so is the set of finite sequences in that set. So since there are countably many sentences, there are also countably many sequents, and there are then also countably many proto-proofs.) Furthermore, it’s pretty straightforward to come up with a systematic method for listing all of the proto-proofs, one by one. Such that eventually, given enough time, each proof will be listed, and nothing besides proofs will be listed. Let list,proto-proofs be a program that does this.
Furthermore, the descriptions we gave of our proof rules are precise enough that it’s easy to systematically check whether or not a given sequence of sequents is really a valid proof or not. Let check-proof be a program that takes a proto-proof as an input and says true if it’s a correct proof, and false otherwise.

The last line of a proof is some sequent \( \Gamma \vdash B \), where \( \Gamma \) is a finite sequence of sentences and \( B \) is a sentence. We can easily check whether or not the conclusion \( B \) is our input sentence \( A \) or not. And also—since the set of axioms is decidable—we can also check whether each element of \( \Gamma \) is an axiom, using check-axiom. Let’s call the program that does both of these things check-conclusion: this takes a proof as input, and returns true if it is a proof of \( A \) from the axioms, and false otherwise.

Now consider the program that runs list-proto-proofs (which spits out proto-proofs) and then runs both check-proof and check-conclusion and returns true if both of those succeed, and false otherwise. Call this program find-proof. If there is a proof of \( A \), then find-proof will eventually reach that proof in its list and return true. But note if there is no proof of \( A \), then find-proof will never find a proof in its list, and it will continue checking and rejecting proto-proofs forever (since there are infinitely many of them). So far, then, we’ve written a program that half solves the problem. We’ve shown that \( X \) is semi-decidable. There’s an effective method for checking that a sentence \( A \) is in \( X \), if \( A \) is in \( X \), but which might fail to ever return a result if \( A \) is not in \( X \).

Finally, since we are assuming that \( X \) is consistent and complete, we know that either \( A \) or \( \neg A \) is in \( X \), and they aren’t both in \( X \). So we can use this trick: we’ll run check-proof on \( A \) and also run check-proof on \( \neg A \) “in parallel”. That is, our final program will run one step of check-proof on \( A \), and then one step of check-proof on \( \neg A \), and then go back and do another step for \( A \), and then another step for \( \neg A \), and so on, back and forth, until one of them says true. We know exactly one of them is going to eventually finish. If we get true from check-proof for \( A \) (meaning that \( A \) is provable from the axioms, and so is a member of \( X \)), our final answer is true. And if we get true from check-proof for \( \neg A \), then our final answer is false.

So, with a little hand-waving, we have shown that if \( X \) has a decidable set of axioms, and \( X \) is consistent and complete, then we can systematically check whether any given sentence is in \( X \). QED.

Now we have everything we need.
**Gödel’s Theorem.** For any theory $X$, if $X$ extends $Q$, and $X$ has a decidable set of axioms, then $X$ is inconsistent or incomplete.

Suppose that $X$ extends $Q$, and $X$ has a decidable set of axioms, and $X$ is consistent and complete. We’ll show that $X$ defines substitution (relationally) and $X$ defines provability—and then we can apply our earlier version of Gödel’s Theorem, and we’re done.

Substitution is easy. We already know that $Q$ defines substitution (relationally), which means there is some formula $\text{sub}[x, y, a]$ such that certain sentences involving this formula are logical consequences of $Q$. Since $X$ extends $Q$, it any consequence of $Q$ is also a consequence of $X$, and so $X$ defines substitution as well.

Now we’ll show $X$ defines provability. We’ll first use the fact that (since $X$ is complete, consistent, and has a decidable set of axioms) $X$ is decidable. Then we’ll use the fact that every decidable set is representable in $Q$. So there is some formula—anticipating what we’ll do with it, we’ll call it $\text{prov}[x]$—such that whenever $q$ denotes $A$,

\begin{align*}
\text{if } A \in X & \text{ then } Q \vdash \text{prov}[q] \\
\text{if } A \notin X & \text{ then } Q \vdash \neg \text{prov}[q]
\end{align*}

(125)

What we want to show is

$$X \vdash \text{prov}[q] \text{ iff } X \vdash A$$

(126)

We’ll do this one direction at a time. First suppose $X \vdash A$. Then (since $X$ is a theory) $A \in X$, so $Q \vdash \text{prov}[q]$. Since $X$ extends $Q$, then, $X \vdash \text{prov}[q]$ as well. For the other direction, suppose it’s not the case that $X \vdash A$. So $A \notin X$, and thus $Q \vdash \neg \text{prov}[q]$, and again since $X$ extends $Q$, $X \vdash \neg \text{prov}[q]$. Then it’s not the case that $X \vdash \text{prov}[q]$, since otherwise $X$ would be inconsistent. This shows that $X$ defines provability. QED.

### 6.9 Consistency

In this section we’ll sketch the proof of Gödel’s Second Incompleteness Theorem: any theory that can prove its own consistency is inconsistent. (Remember, an inconsistent theory can prove anything!)
Suppose we have a theory \( X \) that defines substitution (relationally) and defines provability.

To avoid writing down tons of \( X \)'s, we'll write \( \vdash A \) as an abbreviation for \( X \vdash A \). That is, we’re talking about provability in \( X \), rather than provability from nothing.

Let’s introduce some nice notation from modal logic.

1. **Def.** For any sentence \( A \), let \( \Box A \) be the sentence \( \text{prov } q \) for some term \( q \) that denotes \( A \). (It doesn’t matter which term \( q \), if there’s more than one that denotes \( A \).)

2. **R.** In this notation, what it means for \( X \) to define provability is that

\[
\vdash A \iff \vdash \Box A
\]

(127)

(In the usual terminology from modal logic, the left-to-right direction is called **necessitation**.)

3. The **consistency sentence** for \( X \) is the sentence \( \neg \Box \bot \). Intuitively this says that \( X \) cannot prove a contradiction—that is, \( \Box \bot \) is the **consistency sentence** for the theory \( X \).

4. **R.** We’ll make two further assumptions about the theory \( X \) which I’ll only justify informally.

5. **Assumption.** For any sentences \( A \) and \( B \),

\[
\vdash \Box(A \to B) \to (\Box A \to \Box B)
\]

(128)

Modus ponens is one of our proof rules. So we can take a proof of \( A \to B \) and a proof of \( A \) and chain them together to get a proof of \( B \). This assumption says that \( X \) “knows” this much about how proofs work.

(In the terminology from modal logic, this is called **Axiom K**.)

6. **Assumption.** For any sentence \( A \),

\[
\vdash \Box A \to \Box \Box A
\]

(129)

Note that we already have the fact that if \( \vdash A \), then \( \vdash \Box A \). This is the formalization of that fact in the object language—we’re supposing that not only is the conditional true, but also that \( X \) “knows” it.
6.9 Consistency

(In the terminology from modal logic, this is called **Axiom 4** (or **S4**).)

7. There is a sentence $G$ such that $G \equiv \neg \Box G$.

8. Show:

(a) $\vdash \Box G \rightarrow \Box \neg G$

(b) $\vdash \Box G \rightarrow \Box \bot$

(c) $\vdash \Box \neg \bot \rightarrow \Box G$

9. (Gödel’s Second Incompleteness Theorem.) If $\vdash \neg \Box \bot$ then $\vdash \bot$. That is, if $X$ proves that $X$ is consistent, then $X$ is inconsistent.

10. If $X \vdash \Box A \rightarrow A$ for every sentence $A$, then $X$ is inconsistent. In other words, no consistent theory can prove that it only proves true statements.
1. \( \vdash p \)
   For conditional proof
2. \( \vdash \neg p \)
   For reductio
3. \( p \land \neg p \)
   1 and 2, conjunction intro
4. \( \neg \neg p \)
   2-3, reductio
5. \( p \rightarrow \neg \neg p \)
   1-4, conditional proof