Logic 3

Wolfgang Schwarz

August 2, 2025

Contents

1 Propositional Logic

We are going to introduce formal languages in which one can regiment mathematical and philosophical reasoning, without the distracting complexities and vagaries of natural languages. In this chapter, we begin with the language of propositional logic: the logic of the propositional connectives '$\neg$', '$\to$', '$\land$', '$\lor$', etc. This language is woefully inadequate for any serious applications, but it serves as a valuable prototype to introduce general ideas and techniques that we'll also use for more powerful languages.

After defining the syntax of the propositional language, I'll describe a calculus for reasoning in it. Then I'll give a semantics for the language, which is used to define the concept of validity. Finally, I'll prove that the calculus is "sound" and "complete", meaning that it allows proving all and only the valid sentences.

1.1 Syntax

When talking about a language, it is important to distinguish between the language that is being talked about – called the *object language* – and the *meta-language* in which the talking takes place. Throughout these notes, the meta-language will be English, with some added technical vocabulary that will be introduced as we go along. Our first object language is the language of propositional logic.

In classical propositional logic, all connectives can be defined in terms of '$\neg$' and '$\to$'. For reasons of economy, we will therefore assume that the language we're about to study – our *object language* – has only these two connectives.

Overall, the *primitive expressions* of our propositional language fall into three kinds, whose instances must be distinct:

- a non-empty set of *sentence letters*,
- the *connectives* '$\neg$' and '$\to$',
- the parentheses, '(' and ')'.

The sentence letters as classified as *non-logical* expressions. The other expressions are *logical*. (The point of this classification will become clear in section 1.3.)

Definition 1.1

A *sentence* of $\mathfrak{L}_0$ is a finite string of sentence letters and connectives, built up according to the following formation rules:

(i) Every sentence letter is a sentence.
(ii) If A is a sentence, then so is $\neg A$.
(iii) If A and B are sentences, then so is $(A \to B)$.

A few comments on this definition:

1. I use 'A' and 'B' to stand for arbitrary sentences of $\mathfrak{L}_0$. I will keep using capital letters from the beginning of the alphabet in this fashion, as variables in the metalanguage to denote arbitrary sentences in the object language. I will sometimes use the lowercase letters 'p' and 'q' to denote arbitrary sentence letters.

2. I haven't said what the sentence letters of $\mathfrak{L}_0$ look like. It doesn't matter. Strictly speaking, $\mathfrak{L}_0$ is therefore not a single language, but a family of languages, with different stocks of sentence letters.

3. When I say 'if A is a sentence then $\neg A$ is a sentence', what I mean is that putting '$\neg$' in front of any L_0-sentence yields another L_0-sentence. That is, I use the expression '$\neg A$' to denote the object-language string that consists of the negation symbol followed by whatever string A picks out. Similarly for '$(A \to B)$' and other such cases. It would be tedious to always write: ' '(' followed by A followed by '$\to$' followed by ')' '.

Exercise 1.1

How many sentences are there in $\mathfrak{L}_0$?

Now for some notational shortcuts.

1. We introduce '$\wedge$', '$\vee$', and '$\leftrightarrow$' as metalinguistic abbreviations. That is, if A and B are $\mathfrak{L}_0$-sentences, we write
   - $(A \wedge B)$ for $\neg(A \to \neg B)$;
   - $(A \vee B)$ for $(\neg A \to B)$;
   - $(A \leftrightarrow B)$ for $\neg((A \to B) \to \neg(B \to A))$.

2. It is also convenient to have a zero-ary connective $\top$ that is always true, and its dual $\bot$ that is always false. Where p is an arbitrary sentence letter (say, the alphabetically first), we write
   - $\top$ for $p \to p$;
   - $\bot$ for $\neg(p \to p)$.
3. Where no ambiguity threatens, we'll often omit parentheses. For example, $A \to B$ is short for $(A \to B)$.

To conclude this section on the syntax of propositional logic, I want to introduce (or review) an important method of proof that we're going to use a lot.

Suppose we want to show that every $\mathfrak{L}_0$-sentence has a certain property. Since there are infinitely many sentences, we can't go through the sentences one by one. What we can do instead is this. We first show that all sentence letters have the property. This is called the *base case*. [Hmm, still infinitely many]. Then we show that *if* some sentences have the property *then* so does every sentence built up from them by one of the formation rules in definition 1.1. Since all sentences are built up from sentence letters by the formation rules, it follows that all sentences have the property.

This is called a proof *by induction*. More precisely, it is a proof by induction *on complexity* of formulas because the proof works up from formulas of lowest complexity to formulas of higher complexity.[1]

To illustrate the method, let's prove that every $\mathfrak{L}_0$-sentence has an even number of parentheses.

For the base case, we need to show that every sentence letter has an even number of parentheses. A sentence letter has zero parentheses. Zero is even. So the property holds for sentence letters.

For the inductive step, we need to show that *if* some sentences have an even number of parentheses then so does every sentence generated from these sentences by the formation rules in definition 1.1. We need to consider two cases, because there are two formation rules.

(i) If A has an even number of parentheses, then $\neg A$ has the same even number of parentheses.

(ii) If A and B have an even number mf parentheses, then $(A \to B)$ has that same number plus two, which is still even.

1 This sense of 'induction' is only loosely related to the kind of "inductive reasoning" that is often contrasted with deductive reasoning.

The method of inductive proof can be applied not just to $\mathfrak{L}_0$-sentences. Whenever a set of objects is generated from some base objects by finitely many applications of some operations, we can use the method to show that all of the objects have some property.

Exercise 1.2

Prove by induction on complexity that in any $\mathfrak{L}_0$-sentence A, the initial symbol does not belong to any $\mathfrak{L}_0$-sentence that is a proper part of A. [Bostock pp52f.]

Exercise 1.3

Prove by induction that for every natural number n, $0 + 1 + \ldots + n = \frac{n(n+1)}{2}$. (That is, first show that the claim holds for $n = 0$; then show that if it holds for some n then it also holds for $n + 1$.)

1.2 The propositional calculus

We want to formalize proofs in our language. What is a proof? The traditional answer is: a derivation from some axioms. More precisely, a *proof* of a sentence A is a list of sentences, each of which is either an axiom or follows from [?] earlier sentences in the list by a rule of inference, and whose last element is A.

We'll say that all instances of the following are *logical axioms*.

(A1) $A \to (B \to A)$

(A2) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

(A3) $(\neg A \to \neg B) \to (B \to A)$

Our only rule of inference is detachment or MP. It is again schematic:

(MP) From A and $A \to B$ one may infer B.

The axioms and rules together are called an *axiomatization*. The present axiomatization is due to Lukasiewicz; it is a simplification of Frege's first axiomatization.

Definition 1.2: Proof

A *proof* of a sentence A in the propositional calculus is a finite sequence of sentences $A_1, A_2, \ldots A$, each of which is either an instance of A1–A3 or follows from earlier sentences in the sequence by MP.

We write $\vdash_0 A$ to mean that there is a proof of A.
Example: Here's a proof of p → p:

(Instance of Axiom 1)
$$p \to ((p \to p) \to p)$$
(Instance of Axiom 2)
$$(p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p))$$
(From 1, 2 by MP)
$$(p \to (p \to p)) \to (p \to p)$$
(Instance of Axiom 1)
$$p \to (p \to p)$$
(From 3, 4 by MP)
$$p \to p$$

This result can be generalized. If we replace the sentence letter p by any sentence A, we still get a proof – of $A \to A$ – that conforms to our definition. That is, we've effectively shown that $\vdash_0 A \to A$ for any sentence A.

The example illustrates that our proof technique is hard. It is also unnatural because it focuses on logical truth. More often than not, when we turn to logic, we're interested in *inference*: we want to know whether some conclusion follows from – or can be logically derived from – some premises, which aren't themselves logical truths. In the strict axiomatic method, one shows that some premises $A_1, \ldots A_n$ entail B by proving the conditional $(A_1 \wedge \ldots \wedge A_n) \to B$.

But our concept of proof is naturally generalized to handle derivations from non-logical premises.

Definition 1.3: Deduction

A (propositional) *deduction* of a sentence A from a set Γ (”gamma”) of sentences is a finite sequence of sentences $A_1, A_2, \ldots A$ each of which is either an axiom, an element of Γ, or follows from previous sentences by MP.

We write $\Gamma \vdash_0 A$ to mean that there is a deduction of A from Γ. So $\{p, q\} \vdash_0 p$ is a sentence in our metalanguage saying that p is deducible from the set containing p and q. A proof, as defined above, is a deductions from the empty set of premises.

The following ”structural principles” immediately follow from definition .

(Assumptions)
For all sentences A, $A \vdash A$

(Monotonicity or Thinning)
For all sets of sentences Γ and sentences A, If $\Gamma \vdash A$ then $\Gamma \cup \{B\} \vdash A$

(Cut) For all ... If $\Gamma \vdash A$ and $\Delta, A \vdash B$ then $\Gamma, \Delta \vdash B$

$\Delta \cup \Gamma$ is the *union* of Γ and Δ. (The union of two sets is the set that contains all and only elements that are in either of the sets.) To remove cutter, we will henceforth remove the parentheses and write a comma instead of $\cup$. We’ll also omit the metalinguistic quantifiers over sentences. I.e., we’ll write:

(Assumptions)
$A \vdash A$

(Monotonicity or Thinning)
If $\Gamma \vdash A$ then $\Gamma, B \vdash A$

(Cut) If $\Gamma \vdash A$ and $\Delta, A \vdash B$ then $\Gamma, \Delta \vdash B$

Exercise 1.4

Explain why these principles follow.

The fact that MP can be used in deductions means that we also have the following principle:

(MP*) If $\Gamma \vdash A$ and $\Gamma \vdash A \to B$ then $\Gamma \vdash B$

The following theorem, due to Herbrand (1930), shows that every deduction can be converted into a proof.

Deduction Theorem

If $\Gamma, A \vdash B$ then $\Gamma \vdash A \to B$.

By induction. Machover pp.122f. or Bostock p.203.

I said that the theorem shows that every deduction can be converted into a proof. For suppose there is a deduction of B from A1...An. Then we can move the used premises to the right.

Note that we needed MP and ax.1 and ax.2. With the DT and MP, one can prove ax 1 and ax 2, in the following sense:

xxx prove ax 1

So if we're only interested in whether a deduction or proof exists, we never need to look at A1 and A2 any more.

Exercise 1.5

Show in the same way that A2 is provable. [Bostock 206f.]

Exercise 1.6

The converse of the DT is true as well. [It's equivalent to MP, in fact. Bostock 207]

Now for some facts about negation:

Ex Falso Quodlibet

If Γ ⊢ A and Γ ⊢ $\neg A$, then Γ ⊢ B

1. Γ ⊢ A
2. Γ ⊢ ¬ A
3. $\Gamma, \neg B \vdash \neg A$ (2, Mon)
4. $\Gamma \vdash \neg B \to \neg A$ (3, DT)
5. $\vdash (\neg B \to \neg A) \to (A \to B)$ (A3)
6. $\Gamma \vdash A \to B$ (3,5, MP)
7. Γ ⊢ B (1,6, MP)

Double Negation Elimination

If $\Gamma \vdash \neg\neg A$ then Γ ⊢ A.

1. $\Gamma \vdash \neg\neg A$
2. $\Gamma, \neg A \vdash \neg\neg A$ (1, Mon)
3. $\Gamma, \neg A \vdash \neg A$ (Ass)
4. $\Gamma, \neg A \vdash A$ (1,2, EFQ)
5. $\Gamma \vdash \neg A \to A$ (4, DT)
6. Γ,A ⊢ A (Ass, Mon)

??? Need to get to: 6. $\Gamma \vdash \neg A \to \neg(\neg A \to A)$ See Bostock 209f.

Reductio ad Absurdum

If Γ,A ⊢ B and Γ,A ⊢ $\neg B$, then Γ ⊢ $\neg A$.

1. Γ,A ⊢ B
2. Γ,A ⊢ $\neg B$
3. Γ,A ⊢ $\neg A$ (1, 2, Ex Falso)
4. $\Gamma \vdash A \to \neg A$ (3, DT)
5. $\neg\neg A \vdash \neg\neg A$
6. $\neg\neg A \vdash A$ (5, DNE)
7. $\Gamma \vdash \neg A$ (4,6,MP)

Exercise 1.7

Show that $\Gamma \vdash \bot$ iff there is a sentence A for which $\Gamma \vdash A$ and $\Gamma \vdash \neg A$. [needed in completeness.]

With DT and MP, we never need to look at A1 and A2 any more. With RAA and DNE added, we don't need to invoke A3.

The axioms are somewhat arbitrary. The heart of the propositional calculus consists in the structural deduction principles Assumptions, Monotonicity, Cut, MP, DT, RAA, and DNE.

Exercise 1.8

Show:

(a) $A \to \neg A \vdash \neg A$
(b) $A \vdash \neg\neg A$

Exercise 1.9

Show, by first expanding the definition of $\wedge$:

(a) If $\Gamma \vdash A$ and $\Gamma \vdash B$ then $\Gamma \vdash A \wedge B$
(b) If $\Gamma \vdash A \wedge B$ then $\Gamma \vdash A$ and $\Gamma \vdash B$

The principles we've introduced so far determine *classical* propositional logic. By varying the principles, we can obtain other logics. *Intuitionistic logic*, for example, rejects DNE. It is motivated by a "constructivist" philosophy of mathematics according to which mathematical statements describe the results of mental constructions, rather than an independent realm of mathematical facts. To say that a statement is false, on this view, is to say that there is no way demonstrate its truth. $\neg\neg A$ therefore means that one can't demonstrate that one can't demonstrate A; and this doesn't entail that one can demonstrate A.

A wide class of *substructural logics*, for examples, ...

Relevance logic rejects monotonicity and explosion. The guiding intuition is that the conclusion of a good proof must be genuinely connected to the premises. Monotonicity fails because every premise must be used. Contradictions don't allow deriving arbitrary propositions.

Paraconsistent logic gives up Explosion.

IP proofs are common in maths. But they are a bit strange. If we wan to know why α follows from Γ, finding out that its negation leads to contradiction with Γ is not very informative. We'd like to assume Γ and derive α directly, learning more and more about what Γ worlds look like along the way. This aesthetic point becomes important if we assume that we're not describing an independent reality.

Note that in int logic $\vee$ and $\wedge$ can't be defined in terms of conditional and negation!

...

Return to our main thread.

Our metalinguistic arguments can be seen as a different type of calculus. It is a version of *the sequent calculus*. Here, we use the "shortcuts" not as means to show that there is a Hilbert proof, but as their own set of rules. In this calculus, each line is a *sequent* $\Gamma \Rightarrow B$. To show that B follows from Γ, one tries to derive the sequent $\Gamma \Rightarrow B$. We use the double right arrow instead of the turnstile because the turnstile means that B is deducible from $A_1, \dots, A_n$, In a sequent calculus, one shows $A_1, \dots, A_n \vdash B$ by deriving the sequent $A_1, \dots, A_n \Rightarrow B$.

Each line is either an *Assumption*, of the following form,

(Ass) $\quad \Gamma, A \Rightarrow A$

or follows from previous lines by one of the rules, which are:

(Cut) From $\Gamma \Rightarrow A$ and $\Delta, A \Rightarrow B$ infer $\Gamma, \Delta \Rightarrow B$.

(MP) From $\Gamma \Rightarrow A$ and $\Gamma \Rightarrow A \to B$, infer $\Gamma \Rightarrow B$.

(DT) From $\Gamma, A \Rightarrow B$, infer $\Gamma \Rightarrow A \to B$.

(RAA) From $\Gamma, A \Rightarrow B$ and $\Gamma, A \Rightarrow \neg B$, infer $\Gamma \Rightarrow \neg A$.

(DNE) From $\Gamma \Rightarrow \neg\neg A$, infer $\Gamma \Rightarrow A$.

Note that we no longer get the structural rules of Assumptions, Monotonicity, and Cut for free, as we did when we used the turnstile. The required parts of Assumptions and Monotonicity have been folded into the 'Ass' rule. The Cut rule turns out to be redundant: Gerhard Gentzen (1934) showed that any derivation using Cut can be transformed into a (possibly much longer) cut-free derivation.

To see how this sequent calculus works, here is a simple schematic derivation, showing that $A \to B$ and $B \to C$ together entail $A \to C$.

1.	$A{\to}B,\ B{\to}C,\ A \Rightarrow A{\to}B$	Ass
2.	$A{\to}B,\ B{\to}C,\ A \Rightarrow B{\to}C$	Ass
3.	$A{\to}B,\ B{\to}C,\ A \Rightarrow A$	Ass
5.	$A{\to}B,\ B{\to}C,\ A \Rightarrow B$	MP 1,3
7.	$A{\to}B,\ B{\to}C,\ A \Rightarrow C$	MP 2,5
8.	$A{\to}B,\ B{\to}C \Rightarrow A{\to}C$	DT 7

When writing out this kind of derivation, it becomes tedious to repeat the same assumptions on the left over and over. In introductory logic textbooks, one often finds proofs that look more like this:

Premise

1 $\quad|\ A \to B$

Premise

2 $\quad|\ \underline{B \to C}$

Assumption

3 $\quad|\quad|\ \underline{A}$

$\to$E 1, 3

4 $\quad|\quad|\ B$

$\to$E 2, 4

5 $\quad|\quad|\ C$

$\to$I 3–56 $\quad|\ A \to C$

This is really the same proof, just written in a more convenient style. The correspondence between the two styles becomes apparent if you read the formulas to the left of '$\Rightarrow$' in the sequent derivation as the list of currently undischarged assumptions in the Fitch proof.

This is a form of Natural Deduction. The rules are called "introduction" and "elimination" rules. MP is $\to$E. DT yields $\to$I. RAA is $\neg I$. EFQ is $\neg E$. Full natural deduction systems usually include designated rules for the other connectives, mirroring what you've established in exercise xxx above for the case of $\wedge$.

You may also be familiar with *tableau proofs* or *tree proofs*. These, too, can be seen as variations on sequent proofs, with all sentences pushed to the left of the arrow, and the proofs written upside down in a tree format.

All these techniques are equivalent in the sense that if any of them allows showing that $\Gamma \vdash A$ then so do all the others. For practical applications, natural deduction and tableau proofs are often the most convenient, but – as I said – we'll rarely spell out object-level derivations in the following chapters.

1.3 Semantics

You may have noticed that I have introduced the language L0 without saying anything about what the expressions of the language mean. Introductory logic texts often suggest that '$\neg$' and '$\to$' have (roughly) the same meaning as the English 'not' and 'if ... then'. But we haven't built any such connection to English into the construction of L0. In this chapter, we're going to give a formal theory of meaning for L0.

The status of this kind of theory is philosophically controversial. Some hold that the meaning of a logical expression is given by the rules for reasoning with the expression, and we've already described these rules in the previous chapter. This approach to meaning is sometimes called *inferential role semantics*. (*Semantics* is the study of meaning.)

The kind of theory we're going to describe is a form of *truth-conditional semantics*. The guiding idea of truth-conditional semantics is that the meaning of a sentence can be given by stating what (typically non-linguistic) conditions must be satisfied for the sentence to be true. The German sentence 'Schnee ist weiss', for example, is true iff snow is white, and arguably this information captures the core of its meaning. The meaning of sub-sentential expressions like 'weiss' or '$\neg$' and '$\to$' is given by their contribution to the truth-conditions of sentences in which they occur.

If we apply this approach to L0, we first need to assign truth-conditions to the sentence letters. To a first approximation, this might look as follows:

p : snow is white
q : grass is blue

Here I give the truth-conditions with the help of English sentences. This is not ideal, because English sentences generally don't have fully precise and determinate truth-conditions. It isn't clear what, exactly, must be the case for 'snow is white' to be true. Fortunately, we'll see in a moment that we don't need to worry about this problem because we won't actually need to assign a meaning to the sentence letters at all.

Next, we need to explain how the logical operators contribute to the truth-conditions of sentences in which they occur. This is the important part. We do this inductively.

1. $\neg A$ is true iff A is not true.
2. A$\rightarrow$ B is true iff A is not true or B is true.

To see what this is saying, suppose I had managed to assign precise truth-conditions to the sentence letter p. We would then know, for any conceivable scenario – for any way the world might be – whether p is true in that scenario or not. Call the scenarios in which p is true the *p-scenarios*. The above statement about $\neg$ now tells us $\neg p$ is true in precisely those scenarios in which p is not true. That is, $\neg p$ is true in all the scenarios that aren't p-scenarios. In general, it tells us how to determine the conditions under which $\neg A$ is true based on the conditions under which A is true. Similarly, the statement about $\rightarrow$ tells us how to determine the conditions under which $A \rightarrow B$ is true based on the conditions under which A and B are true.

A consequence of our rules is that $\neg\neg p$ has the same meaning as p, for it is true under the exact same conditions (in all and only the p-scenarios). With a little effort, you can check that the same holds for $(p \rightarrow q) \rightarrow p$: this, too, is true in all and only the p-scenarios.

The truth-conditional conception of meanings is useful in logic because it ties in with a natural conception of entailment. Intuitively, some premises entail a conclusion iff the truth of the premises guarantee the truth of the conclusion; that is, if there is no conceivable scenario in which the premises are true while the conclusion is false. If that's right then all we need to know to determine whether the premises entail the conclusions is their truth-conditions.

In fact, logic is about a particular type of entailment, which we call *logical entailment*. For example, suppose we give the following truth-conditions to p and q:

p: Snow is white. q: Snow is green.

Then p entails $\neg q$. (There is no conceivable scenario in which snow is white while it is not the case that snow is not green.) But the inference from p to $\neg q$ is not logically valid.

Why not? Because it depends on the meaning of the non-logical expressions p and q. Logic abstracts away from the interpretation of the non-logical expressions. Some premises *logically entail* a conclusion iff there's no conceivable scenario in which premises are true and conclusion false, on any interpretation of the non-logical expressions.

This obviously requires a distinction between ”logical” and ”non-logical” expressions. There was once a debate over which expressions in English belong into which group, but it is best not to think of this as an absolute distinction. In *epistemic logic*, for example, a regimented concept of 'it is known that' counts as logical. Since propositional logic is the logic of the truth-functional connectives, we count ¬ and → as logical and the sentence letters as non-logical.

It's because propositional logic abstracts away from the meaning of the sentence letters that we don't need to worry about how to assign them a particular meaning. We can leave them uninterpreted.

Our definition of logical entailment quantifies over all conceivable scenarios and all interpretations of the sentence letters: $A_1, \dots A_n$ entail B iff every scenario and interpretation that makes $A_1, \dots A_n$ true also makes B true. We can make this much simpler, and also more precise.

Think of what you need to know about a pair of a scenario S and an interpretation I of the sentence letters to determine whether an arbitrary L0-sentence – say, $\neg p$ – is true. I could tell you that p means that snow is blue, and that the scenario is one in which snow is red. You could then figure out that $\neg p$ is true, relative to S and I. But you don't need all that information. It would be enough if I merely told you that p means something that isn't true in S. By the interpretation rule for negation, this is enough to determine that $\neg p$ is true in S under the interpretation I. Generalizing, all the information we need about a pair of a scenario S and an interpretation I to determine whether an arbitrary L0-sentence is true in S under I is which sentence letters are true and which are false in S under I. This means that instead of quantifying over scenarios and interpretations, we can simply quantify over assignments of truth-values to the sentence letters.

Definition 1.4: Model

A *model* for L0 is an assignment σ ("sigma") of truth-values to the sentence letters of L0.

That is, a model is a function that assigns to each sentence letter one of the two truth values, which I'll label 'T' (true) and 'F' (false).

We extend this assignment to all sentences in accordance with our above interpretation rules for '¬' and '→'. We write σ ⊨ A (read: σ satisfies A, σ models A, or p is true under σ) to mean that A is true in the model A. It is defined as follows:

Definition 1.5: Satisfaction

For any sentence letter p and sentences A and B:

1. $\sigma \models p$ iff $\sigma(p) = T$.
2. $\sigma \models \neg A$ iff $\sigma \not\models A$.
3. $\sigma \models A \to B$ iff $\sigma \not\models A$ or $\sigma \models B$.

Finally, we can define logical entailment in terms of models:

Definition 1.6: Logical Entailment

A set of sentences Γ entails a sentence A, written $\Gamma \models$ A, iff for every model σ, if σ satisfies every sentence in Γ, then σ satisfies A.

Exercise 1.10

Explain why $\Gamma \models$ A iff there is no conceivable scenario in which every sentence in Γ is true while A is false, on any interpretation of the sentence letters.

Exercise 1.11

Show that A is valid iff A is entailed by $\emptyset$.

As in the case of the single-barred turnstile, we often drop the curly braces and use a comma instead of '$\cup$'.

We allow Γ to be infinite, and to be empty. If something is entailed by the empty set of premises, it is *valid*.

Definition 1.7: Validity

A is *valid* if $\models$ A, i.e., if it is satisfied by every truth-assignment.

Exercise 1.12

Which of these claims are true: (a) $\models \top$, (b) $\models \bot$, (c) $\top \models \bot$, (d) $\bot \models \top$.

Exercise 1.13

Show that A,¬ B ⊨ ⊥ iff A ⊨ B.

Warning (or exercise?): We now have four arrow-like symbols: ⊢, ⊨, ⊨, and →. (We also have the ⇒ symbol mentioned in discussion of the sequent calculus, but we'll never use that again.) Make sure you can tell the difference!

Of course there's more to meaning than truth-value. But the operators ¬ and → are truth-functional, so any further difference doesn't affect whether an L_0 sentence is true in a scenario.

We need more fine-grained models if we add non-truth-functional operators, such as an operator that regiments a sense of 'necessarily'. Intuitionistic logic interprets ¬ and → in a non-truth-functional way.

The standard models of modal logic have possible worlds. One also adds some axioms for how behaves.

At this point I could mention that intuitionistic logic requires a different kind of interpretation. What could ¬ mean so that ¬¬p doesn't entail p, or so that LEM fails? In intuitionistic logic, we don't assume we're describing a fixed reality. ¬p means that one can refute p. p→q means that one can derive q from p.

1.4 Soundness and Completeness

In the last two sections, we have explored two very different perspectives on logic, called *proof-theoretic* and *model-theoretic*. The central concept of proof theory is ⊢. Here we study whether some sentences can be proved or derived from others, without caring about what the sentences mean or what it would take for them to be true. The sentences are simply arrangements of symbols, and proofs are manipulations of such arrangements symbols. In model theory, by contrast, we think of formal sentences as describing some putative aspect of reality, although we generally don't fix a particular interpretation. The central concepts of model theory are ⊨ and ⊨.

Ideally, we'd like the two perspectives to harmonize, so that $\Gamma \vdash A$ iff $\Gamma \models A$. That is, a sentence is deducible from some premises iff it is entailed by the premises. This is by no means obvious. We'll show that it is true.

We have two directions to show. We first show that if Γ ⊢ A, then Γ ⊨ A. This shows that the calculus is *sound* with respect to our model-theoretic conception of entailment and validity: anything that can be proved is valid, and anything that can be deduced from some premises is entailed by the premises.

Then we show that if Γ ⊨ A, then Γ ⊢ A. This shows that the calculus is *complete*: anything that is valid can be proved, and anything that is entailed by some premises can be deduced from the premises.

The soundness proof is straightforward. Assume there is a sequence $A_1, A_2, \ldots, A_n$ such that $A_n = A$ and each A_i is either an axiom, a member of Γ, or follows from previous sentences by MP. We prove by induction on k that $\Gamma \models A_k$ for all k from 1 to n.

1. If A_i is an axiom, then it is satisfied by every truth-assignment σ. [Exercise?]
2. If A_i is a member of Γ, then obviously it is satisfied by every truth-assignment σ that satisfies Γ.
3. If A_i follows from previous sentences by MP, then there are sentences A_j and A_k such that $A_i = A_j \to A_k$. By the induction hypothesis, both A_j and A_k are satisfied by σ. Since σ satisfies A_j, it must also satisfy A_i, because $\sigma \models A_j \to A_k$ iff $\sigma \not\models A_j$ or $\sigma \models A_k$.

Exercise 1.14

We might wonder whether a given proof system is consistent. Explain why soundness provides an answer. (This shows that model theory can be useful for proof theory.)

The first published proof of the truth-functional completeness of an axiomatic system was due to the American logician Emil Post (1921).

Next, completeness. This is harder. It was first established by Paul Bernays in 1918. We're going to use a later technique due to Leon Henkin (1949) that we'll re-use in chapter 3 to prove completeness for first-order logic.

Before we start, we need to two key concepts for Henkin's proof.

Definition 1.8: Consistency

A set of sentences Γ is *consistent* if one cannot derive a sentence and its negation from it.

Strictly speaking, consistency is relative to a calculus.

Definition 1.9: Satisfiability

A set of sentences Γ is *satisfiable* if there is a model σ that satisfies every sentence in Γ.

Exercise 1.15

Show that Γ is consistent iff (a) there is some sentence that cannot be derived from it; (b) one cannot derive $\bot$ from it.

Exercise 1.16

Show that $\Gamma \cup \{\neg A\}$ is satisfiable iff $\Gamma \not\models A$.

Now we have to show that if $\Gamma \models A$, then $\Gamma \vdash A$. The proof is by contraposition. That is, we assume $\Gamma \nvdash A$ and show that, in any such case, $\Gamma \not\models A$. The following lemma allows us to reformulate this task.

Lemma 1.1

$\Gamma \nvdash A$ iff $\Gamma \cup \{\neg A\}$ is consistent.

If $\Gamma \cup \{\neg A\}$ were inconsistent, we could deduce A from Γ by PIP. Contraposing, this means that if $\Gamma \nvdash A$ then $\Gamma \cup \{\neg A\}$ is consistent.

Conversely, if $\Gamma \vdash A$ then $\Gamma, \neg A \vdash A$ by Monotonicity, and $\Gamma, \neg A \vdash \neg A$ by Assumptions; so $\Gamma \cup \{\neg A\}$ is inconsistent. □

Now suppose we can show that any consistent set of sentences is satisfiable. Since $\Gamma \cup \{\neg A\}$ is consistent, it will follow that it is satisfiable. This means that $\Gamma \not\models A$. So all we need to show is that every consistent set of sentences is satisfiable.

To this end, we first show that $\Gamma \cup \{\neg A\}$ can be extended to a *maximal consistent* set that contains, for each sentence, either that sentence or its negation. Then we show that every maximally consistent set is satisfied by the truth-assignment that assigns 1 to every sentence letter in the set and 0 to every sentence letter not in the set.

En route to the first step, we start with an easy observation.

Extension Consistency

If a set Γ is consistent, then for any sentence A, either $\Gamma \cup \{A\}$ or $\Gamma \cup \{\neg A\}$ is consistent.

Let Γ be any consistent set and A any sentence.

That $\Gamma \cup \{A\}$ is inconsistent means there are sentences $A_1, \dots, A_n$ in Γ such that $A_1, \dots, A_n, A \vdash \bot$. By reductio, it would follow that $A_1, \dots, A_n \vdash \neg A$.

That $\Gamma \cup \{\neg A\}$ is inconsistent means that there are sentences $B_1, \dots, B_m$ in Γ such that $B_1, \dots, B_m, \neg A \vdash \bot$. By reductio, it would follow that $B_1, \dots, B_m \vdash \neg\neg A$.

If both $\Gamma \cup \{A\}$ and $\Gamma \cup \{\neg A\}$ were inconsistent, it would follow by monotonicity that there are sentences $A_1, \dots, A_n, B_1, \dots B_m$ in Γ from which one can infer both $\neg A$ and $\neg\neg A$. By exercise above, this means that Γ is inconsistent.

(Exercise: $\neg A, A \vdash \bot$, by ex falso?)

Now the first step:

Lindenbaum's Lemma

Every consistent set is a subset of some maximal consistent set.

Let S_0 be some consistent set of sentences. Let $A_1, A_2, \dots$ be a list of all sentences in some arbitrary order. For every number $i \geq 0$, define

$$S_{i+1} = \begin{cases} S_i \cup \{A_i\} & \text{if } S_i \cup \{A_i\} \text{ is consistent} \\ S_i \cup \{\neg A_i\} & \text{otherwise.} \end{cases}$$

This gives us an infinite list of sets $S_0, S_1, S_2, \dots$. Each set in the list is consistent: S_0 is consistent by assumption. And if some set S_i in the list is consistent, then either $S_i \cup \{A_i\}$ is consistent, in which case $S_{i+1} = S_i \cup \{A_i\}$ is consistent, or $S_i \cup \{A_i\}$ is not consistent, in which case S_{i+1} is $S_i \cup \{\neg A_i\}$, which is consistent by observation **??**. So if any set in the list is consistent, then the next set in the list is also consistent. It follows that $S_0, S_1, S_2, \dots$ are all consistent.

Now let S be the set of sentences that occur in at least one of the sets $S_0, S_1, S_2, S_3 \dots$. (That is, let S be the union of $S_0, S_1, S_2, S_3, \dots$.) Evidently, S_0 is a subset of S. And S is maximal. Moreover, S is consistent. For if S were not consistent, then it would contain some sentences $B_1, \dots, B_n$ from which $\bot$ is provable. All of these sentences would have to occur somewhere on the list $A_1, A_2, \ldots$. Let A_j be a sentence from $A_1, A_2, \ldots$ that occurs after all the $B_1, \ldots, B_n$. If $B_1, \ldots, B_n$ are in S, they would have to be in S_j already, so S_j would be inconsistent. But we've seen that all of $S_0, S_1, S_2, \ldots$ are consistent. □

Note that the only axioms we needed to prove Lindenbaum's Lemma were whatever was needed to prove Reductio, for observation **??**.

We also assumed that there's an enumeration of all sentences. How could that be? Zig-zag.

The lemma actually still holds if there's no enumeration, but it then requires AoC.

Now we show that every maximal consistent set is satisfied by a truth-assignment that assigns 1 to every sentence letter in the set and 0 to every sentence letter not in the set.

Truth Lemma

Every maximal consistent set is satisfied by a truth-assignment that assigns 1 to every sentence letter in the set and 0 to every sentence letter not in the set.

Let S be a maximal consistent set. Define a truth-assignment σ as follows: for every sentence letter p, $\sigma \models p$ iff $p \in S$. We show that σ satisfies every sentence in S by induction on the complexity of sentences.

1. A is a sentence letter. Then $A \in S$ iff $\sigma \models A$ by definition.
2. A is a $\neg B$. If $A \in S$ then $B \notin S$ by consistency, and by induction hypothesis, $\sigma \not\models B$. By truth definition, $\sigma \models A$. Conversely, if $A \notin S$ then $B \in S$ by maximality, and by induction hypothesis, $\sigma \models B$. By truth definition, $\sigma \not\models A$.
3. A is $B \to C$. Assume $A \in S$. If B in S then by MP so is C. Both are true at sigma by i.h. A is true by def truth. If B not in S then by i.h. B is false at sigma and so A is true at sigma by def truth.

Assume conversely that $B \to C$ true at sigma By truth def, either B is false at sigma or C is true at sigma. Let's go through both cases.

If B is false at sigma, by i.h. and maximality, $\neg B$ in S. But $\neg B \vdash B \to C$. So $B \to C$ in sigma. (Why $\neg B \vdash B \to C$? We have $\neg B, B \vdash \bot$; so $\neg B, B \vdash C$ by ex falso; so $\neg B \vdash B \to C$ by DT.)

If C is true at sigma, by i.h. C in S. And $C \vdash B \to C$. (Why? Well, $C, B \vdash C$, so by DT..)

Corollary: $\vdash$ is *consistent*, meaning that $\bot$ is not derivable.

Exercise 1.17

Show that if we add to the axioms (A1)-(A3) any axiom schema that is not already provable in our calculus, we get an inconsistent calculus. (This means that our axiomatization is *Post-complete*, after Emil Post, who first proved the present fact in 1921.) Hint: By the completeness theorem, any schema that isn't provable in our calculus has invalid instances. Can you see why a schema with invalid instances must have inconsistent instances?

[Answer, I guess: take an instance that is false under some valuation; now replace all sentence letters in the instance that are false under that valuation by $\bot$ and all true ones by $\top$; the result is still an instance, and it is false under every valuation. By completeness, it is refutable.]

2 First-Order Predicate Logic

2.1 Syntax

In this chapter, we'll review the syntax and semantics of first-order predicate logic. Unlike with propositional logic, most – some would even say: all – mathematical reasoning can be formalized in this language.

I'll begin with a simple version the language, without function symbols and identity. These will be added in section 2.4.

For the moment, then, the *primitive expressions* of a first-order language $\mathfrak{L}_1$ fall into the following kinds, whose instances must be distinct:

- an infinite set of *individual variables*,
- an infinite set of *individual constants*,
- for each natural number n, a set of *n-ary predicate symbols*,
- the *connectives* '$\neg$' and '$\to$',
- the *universal quantifier symbol* '$\forall$',
- the parentheses '(' and ')'.

The individual constants and predicate symbols are classified as non-logical.

Definition 2.1

The *singular terms* of a first-order language $\mathfrak{L}_1$ are its individual constants and its variables.

A *formula* of $\mathfrak{L}_1$ [later use $\mathfrak{L}_1^=$] is a finite string built up according to the following formation rules:

(i) If P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are singular terms then $Pt_1 \ldots t_n$ is a formula.
(ii) If A is a formula, then so is $\neg A$.
(iii) If A and B are formulas, then so is $(A \to B)$.
(iv) If x is a variable and A is a formula then $\forall xA$ is a formula.

Here, 'P', 't_1', 't_n', 'A', 'B', 'x' are metalinguistic variables standing for expressions in the object language $\mathfrak{L}_1$. I haven't specified what the predicate symbols, individual constants, and variables of $\mathfrak{L}_1$ should look like. A popular convention is to use capital letters as predicate symbols, lower-case letters from the beginning of the alphabet as individual constants, and lower-case letters from the end of the alphabet (perhaps with numerical subscripts) as variables. With this convention, the following might be example formulas, assuming that 'F' is 1-ary (also called *monadic*) and 'R' 2-ary (or *binary*):

$$Fa,\;\; Rab,\;\; Rax,\;\; (\neg Fy \to Rax),\;\; (Fa \to \forall x(Fx \to Rax))$$

Exercise 2.1

Is $\forall x Fa$ a formula of $\mathfrak{L}_1$?

As in the case of propositional logic, we introduce some shortcuts in the metalanguage, writing

- $(A \land B)$ for $\neg(A \to \neg B)$;
- $(A \lor B)$ for $(\neg A \to B)$;
- $(A \leftrightarrow B)$ for $\neg((A \to B) \to \neg(B \to A))$.
- $\top$ for $A \to A$;
- $\bot$ for $\neg(A \to A)$;
- $\exists x A$ for $\neg\forall x \neg A$.

We'll omit parentheses when they are not needed for disambiguation.

Now a few words on how this kind of language is meant to be used.

Individual constants function as names. Each individual constant picks out an object – which might be a person, a number, a set, or whatever. Predicate letters express properties or relations. In a formal theory of arithmetic, for example, we might have individual constants '0' and '1' for the numbers 0 and 1, and a binary predicate 'G' that expresses the greater-than relation. '$G10$' is then a true formula, stating that 1 is greater than 0. '$G11$' falsely states that 1 is greater than itself.

The quantifier symbol '$\forall$' allows making general claims about all objects – or, more precisely, about all objects in the intended domain of discourse. In a theory of arithmetic, the intended domain of discourse would consist of the natural numbers 0, 1, 2, 3, etc. It would not include, say, Julius Caesar. Thus '$\forall x Gx0$ 'might state (falsely) that every natural number is greater than 0, while '$\forall x(Gx1 \to Gx0)$' states (truly) that every number that is greater than 1 is greater than 0. '$\exists x G1x$', which is short for '$\neg\forall x \neg G1x$', states (truly) that 1 is greater than some number.

Exercise 2.2

I said above that the primitive expressions must be "distinct". To be pediantic, we should say that none is part of another. Explain why it would be a problem to have '1', '2', ...

The quantificational constructions of first-order logic don't work like ordinary quantificational constructions in natural language, and it takes some practice to become fluent in their use. The closest translation of '$\forall x Gx0$' in English is something like

> Everything is such that it is greater than 0.

Of course, this can be simplified to 'everything is greater than 0', but here 'everything' combines directly with a predicate ('is greater than 0'), whereas the $\mathfrak{L}_1$-quantifier '$forallx$' combines with an expression of sentential type, with '$Gx0$', which means something like 'it is greater than 0'. The bound variable 'x' appears in the English translation only once, as the pronoun 'it'. Overt variables are rarely used in English, but they can be useful when quantifiers are nested:

> For every number x there is a number y greater than x such that every number greater than y is greater than x.

This can be easily expressed in $\mathfrak{L}_1$:

$$\forall x \exists y (Gyx \land \forall z (Gzy \to Gzx)).$$

[Exercise?]
Now consider the expression

$$Gx0.$$

According to definition **??**, it is a formula. But what does it say? I'm not asking how to pronounce it. We can, of course, read it as 'gee ex zero' or as 'x is greater than zero'. But 'x' doesn't pick out any definite number. Only individual constants pick out definite objects. Variables are devices to construct quantified statements. So '$Gx0$' doesn't really say anything. It is neither true nor false. (We might say that it is true "relative to some interpretations of 'x' " and false relative to others. For example, it is true if we interpret 'x' as picking out 1, and false if we interpret it as picking out 0.)

The problem with '$Gx0$' is not that it contains a variable. '$\forall xGx0$' also contains a variable, but it raises no problems. On out arithmetical interpretation, it says that every number is greater than 0, which happens to be false. The problem with '$Gx0$' is that it contains a *free* variable – a variable that isn't bound by a quantifier.

Formulas with free variables are called *open*. Formulas with out free variables are *closed*. Only closed formulas make an outright claim about the intended domain of discourse. We allow for open formulas only because they allow for a simple specification of the construction rules for larger sentences.

Let's make these distinction more precise.

Definition 2.2

A *subformula* of a formula is a part of a formula that is itself a formula.
A *quantifier* consists of the symbol '$\forall$' followed by a variable. The variable is said to be *bound* by the quantifier.
The *scope* of an occurrence of a quantifier $\forall x$ in a formula is the shortest subformula that contains the occurrence.
An occurrence of a variable in a formula is *bound* if it lies in the scope of an occurrence of a quantifier that binds it.
An occurrence of a variable that isn't bound is *free*.
A formula in which some variable occurs free is *open*.
A formula in which no variable occurs free is *closed*.
A *sentence* is a closed formula.

Exercise 2.3

Why do I say that "occurrences" of a variable in a formula are free or bound? Why not simply say that a variable is free or bound?

Exercise 2.4

Which of these are open/closed? ...

Exercise 2.5

$\mathfrak{L}_1$ *extends* $\mathfrak{L}_0$ in the sense that every $\mathfrak{L}_0$-sentence is also an $\mathfrak{L}_1$-sentence. Explain why this is true.

The language of first-order logic is more restrictive than the formal language introduced by Frege. We only allow quantifying into the position of individual constants. Frege also allowed quantification into the position of predicates, as in $\forall X X a$. This kind of *second-order* quantification has interesting applications in mathematics and philosophy, but we'll only turn to it in chapter **??**.

2.2 The first-order predicate calculus

We're going to define a Hilbert-style proof calculus for $\mathfrak{L}_1$. We'll have the same axiom schemas A1-A3 as in the propositional calculus, and the same inference rule of Modus Ponens:

(A1)	$A \to (B \to A)$
(A2)	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
(A3)	$(\neg A \to \neg B) \to (B \to A)$
(MP)	From A and $A \to B$ one may infer B.

In addition, we need some principles for dealing with quantifiers. For example, we want to allow reasoning from the universal statement $\forall x F x$ to its instances like Fa and Fb. To state this in full generality, we need to define the notion of substitution.

Definition 2.3

If A is an $\mathfrak{L}_1$-formula, x a variable, and t a singular term, then $A(x/t)$ is the formula obtained from A by replacing all free occurrences of x in A with t.

The substitution operator '(x/t)' obviously belongs to our metalanguage, not to $\mathfrak{L}_1$ itself. For example, the metalinguistic expression '$Fx(x/a)$' denotes the object-language sentence 'Fa'. In informal discussions, I'll sometimes also write '$A(x)$' to indicate that A is a formula in which the variable x occurs freely, in which case '$A(t)$' is shorthand for '$A(x/t)$'.

Exercise 2.6

Let A be $\forall x(Fx \to Gy) \to \forall y Fy$. What is $A(x/a)$?

The general rule of "Universal Instantiation" says that one may reason from $\forall xA$ to $A(x/c)$, where c is an individual constant. In Hilbert-style axiomatic calculi, we try to minimize primitive rules. So instead of adding a new inference rule, we'll add a corresponding axiom schema:

(A4) $\quad \forall xA \to A(x/c)$.

With this, we can reason from $\forall xA$ to $A(x/c)$ by Modus Ponens.

We'll introduce a genuine rule – called *(Universal) Generalization* – for the converse inference:

(Gen) $\quad$ From A one may infer $\forall xA(t/x)$.

This rule is obviously unsound when reasoning from assumptions: from the assumption that Joan sleeps we can't infer that everyone sleeps. But remember that each line in a (Hilbert-style) axiomatic proof is either an axiom or follows from an axiom by an inference rule. None of our axioms (or rules) will make specific claims about a particular object c that it wouldn't make about all objects. Thus if $A(c)$ can be derived from the axioms then one could equally derive $A(d)$ for any other constant d.

To get a complete calculus, we need one more axiom:

(A5) $\quad \forall x(A \to B) \to (A \to \forall xB)$, if x is not free in A.

To see the point of this, suppose that in the course of a proof we have established the following claims, for some A and $B(x)$, where x isn't free in A:

$\forall x(A \to B(x))$
A

If we had a rule of Universal Instantiation, we could deduce $A \to B(c)$ from the first line, then use Modus Ponens to infer $B(c)$ and finally infer $\forall xB(x)$ by Universal Generalization. It turns out that this reasoning can't be replicated after we've replace Universal

Instantiation by the axiom schema (A4). So we add (A5), which allows inferring $\forall x B(x)$ by two applications of Modus Ponens.

Definition 2.4: Proof

A *proof* of a sentence A in the basic first-order predicate calculus is a finite sequence of sentences $A_1, A_2, \ldots A$, each of which is either an instance of A1-A5 or follows from earlier sentences in the sequence by MP or Gen.

I'll use '$\vdash A$' to express that A is provable in the first-order predicate calculus.

As in the case of propositional logic, it's convenient to have a more general notion of proof that covers deductions from premises.

Definition 2.5

A (first-order) *deduction* from a set Γ of sentences is a finite list of sentences each of which is either an axiom, an element of Γ, or follows from previous sentences by MP or Gen – but without applying Gen to individual constants c that occur in Γ.

A deduction *of* a sentence A from a set Γ of sentences is a deduction from Γ whose last element is A.

I've already mentioned that the Gen rule is not generally sound for deductions from premises: we don't want to infer $\forall x Fx$ from Fa. In a deduction, the individual constant c that is turned into a universally bound variable must not occur in any premise.

Let's show that, as in the propositional calculus, the notion of deduction reduces to our original notion of proof:

Deduction Theorem

If $\Gamma, A \vdash B$ then $\Gamma \vdash A \to B$.

The proof is almost exactly like in the previous chapter. We only need an extra clause for applications of the rule (Gen), where we appeal to the axiom schema A5.

See e.g. Bostock 224

Given Gen and the DT, we can derive A5, so we won't need to look at that again. (Bostock 224f., probably best to use the proof from p.225)

From A4 and the DT, we obviously get the rule of Universal Instantiation, or ∀ Elimination:

Universal Instantiation
f $\Gamma \vdash \forall x A$ then $\Gamma \vdash A(x/c)$ for any individual constant c.

Now we don't need A4 any more.

Since we have the DT, the proofs of Reductio, Ex Falso, DNE, and PIP from the previous chapter all go through unchanged. We also have the structural principles Assumptions, Monotonicity, and Cut.

In fact, we know from completeness theorem that all truth-functional tautologies are provable from P1-P3 and MP. So we have this fact:

$\Gamma \vdash A$ whenever A is a tautology.

This could be considered a derived rule. It's an odd rule. But it's verifiable: there is a mechanical way to check if something is a tautology; e.g. with truth tables.

In the previous chapter, I should that our results about deduction can be turned into different types of calculi. This is still true for first-order logic. I won't stop to explain it again.

Exercise: show that if $\Gamma \vdash A(x/c)$ then $\Gamma \vdash \exists x A$.

Side note: I have stipulated that each line in a proof must be a sentence. In some versions of the first-order calculus, open formulas are allowed in proofs: from $\forall x F x$, one may infer Fx; the variable indicates that x is an "arbitrary individual". The same *sentences* are provable in either version of the calculus. Since individual constants aren't used in reasoning from universal formulas, the alternative calculus doesn't need my assumption that $\mathfrak{L}_1$ has an unbounded supply of such constants. End of side note.

2.3 Semantics

A language is not just a system of meaningless sounds and scribbles, combined according to seemingly arbitrary syntactic rules. If we want to use a first-order language $\mathfrak{L}_1$ to formalize statements about this or that subject matter, we need to give meaning to its non-logical vocabulary.

On the truth-conditional approach, sentence meanings are understood in terms of truth-conditions: the meaning of a sentence should settle, for each conceivable scenario, whether the sentence is true or false (in that scenario). The meaning of sub-sentential expressions is determined by their contribution to sentence meaning. Let's think about what this implies for the expressions of $\mathfrak{L}_1$, focusing on simple atomic sentences like 'Fa'.

These are meant to function much like subject-predicate sentences in natural language: 'Fa' might be a formal way of saying 'Kurt Gödel is a logician'. In the English sentence, the name 'Kurt Gödel' picks out an individual, and the predicate 'is a logician' attributes a property to that individual. This suggests that we could fix the truth-conditions of atomic $\mathfrak{L}_1$-sentences by assigning

- a (definite, precise) object to each individual constant, and - a (definite, precise) property or relation to each predicate letter.

We don't need to worry much about the concept of an "object". We'll assume that in any use of a first-order language, there is an intended domain of things we want to talk about; each such thing will qualify as an "object" – as a candidate denotation of individual constants. But what is a property (or relation)?

The standard approach in the model theory of first-order logic is to represent a property by its *extension*: by the sets of objects that have the property. The property of being a logician, for example, would be identified with the set { Kurt Gödel, Gottlob Frege, Rózsa Péter, …} of all logicians.

As a general theory of meaning, this is clearly inadequate. After all, there are conceivable scenarios in which Kurt Gödel is not a logician, or in which, say, Carla Bruni is a logician; but there is no conceivable scenario in which Gödel isn't in the set $\{KurtGdel, ...\}$ or in which Carla Bruni is in that set. There are two reasons why we can nonetheless largely get away with the "extensional" account of predicate meaning.

The first is that many logicians are mostly interested in mathematical interpretations, where we might read 'a' as denoting the number 2 and 'F' as expressing the property of being prime. While Kurt Gödel could have failed to be a logician, the number 2 could not have failed to be prime: there is no conceivable scenario in which 2 isn't prime; nor is there a conceivable scenario in which, say, the number 4 is prime. In general, it is reasonable to hold that purely mathematical truths don't vary across conceivable scenarios. In this special case, truth-conditions are determined by extensions.

The second reason why we can get away with an extensional construal of properties is analogous to the reason why we can get away with modelling sentence meanings as truth-values in propositional logic.

Recall our intuitive gloss on logical entailment: some premises logically entail some conclusion if there is no conceivable scenario in which the premises are true and the conclusion false, under any interpretation of the non-logical vocabulary; equivalently: – if there is no pair of a scenario and an interpretation that makes the premises true and the conclusion false. Now, what do you need to know about a scenario S and an interpretation I in order to figure out whether an $\mathfrak{L}_1$-sentence like Fa is true or false (in S under I)?

It would suffice to know (1) what object is picked out by 'a' under I, (2) what property is expressed by 'F' under I, and (3) whether that object has that property in S. But you don't need all that information. It would also suffice to know (1) what object is picked out by 'a' under I, and (2) which objects in S have the property expressed by 'F' under I. For example, if I tell you that 'a' picks out Kurt Gödel and that 'F' expresses a property that belongs to Kurt Gödel and Carla Bruni (in S), you'll know enough to figure out that 'Fa' is true – although you don't really know what the sentence says. That is, if we're interested in interpretations only for the purpose of developing a precise account of (first-order) logical entailment, we can ignore any aspect of predicate meaning that goes beyond extension.

Definition 2.6

A *model* M of a first-order language $\mathfrak{L}_1$ consists of

(i) a non-empty set D, called the *domain* or *universe* of M, and
(ii) a function $\imath$ that assigns
- to each individual constant of $\mathfrak{L}_1$ an object in D, and
- to each n-ary predicate of $\mathfrak{L}_1$ a set of n-tuples of objects in D.

(Models are often called 'interpretations', but outside pure mathematics this is a dubious label, for the reasons given above.)

Again, the domain of a model is assumed to comprise all the objects one intends to speak about. The domain can be infinite, but it can't be empty. Each individual constant is associated with an object in the domain. Different constants can pick out the same object. (We need at least one object in the domain, as otherwise we couldn't interpret the names.) Each predicate symbol is associated with an n-tuple of objects from the domain. An *n-tuple* is a list of n objects. A 1-tuple is simply an object. So a "set of 1-tuples of objects" is simply a set of objects; a set of 2-tuples of objects is a set of pairs of objects, and so on.

Note that it is not obvious whether every model represents a genuinely possible interpretation and scenario. There are lots of infinite sets of numbers, for example, many of which are not specifiable in English in any finite way. Could each of them possibly be the meaning of a predicate? We could restrict the possible interpretations of predicates to some subsets of the domain. But to which? Anyway, it makes no difference to the logic. We'll return to this in the case of 2nd-order logic though. (There, if we leave open the higher-order domain, only saying that it's some subset of the powerset of the domain, the class of models becomes much larger. As a result, the logic becomes much weaker.)

Exercise: what about zero-ary predicates? can you make this work?

Now we can define what it takes for a sentence A to be true in a model M – for short, $M \models A$. Most of this is easy. For example, an atomic sentence Fa is true in a model M iff the object assigned (by M) to 'a' belongs to the set assigned to 'F'. The clauses for $\neg$ and $\to$ remain as in the previous section. The only difficulty arises with quantified sentences. Let $A(x)$ be some formula in which x is free. Under what conditions is $\forall x A(x)$ true in a model M?

As a first shot, we might suggest that $\forall x A(x)$ is true iff $A(c)$ is true for every individual constant c. (This is called a *substitutional interpretation* of the quantifier.) But what if some objects in the intended domain don't have a name? Most real numbers, for example, are not denoted by any English expression (because there are strictly more real numbers than English expressions), and we generally want to allow for this kind of situation in $\mathfrak{L}_1$

There are two common ways to proceed. The first is to declare that $\forall x A(x)$ is true in a model M iff $A(x)$ is true *for every way of assigning an individual to x*. This requires extending the definition of truth in a model so that open formulas are true or false in a model M relative to an *assignment function g* that assigns an object to each variable.

The second option is to declare that $\forall x A(x)$ is true in M iff $A(c)$ is true in every variant of the model M that differs from M at most in the object it assigns to c, where c is some individual constant that doesn't occur in $A(x)$.

The two options amount to the same thing. I'll use the second. Here, then, is the full definition of truth in a model.

Definition 2.7

An $\mathfrak{L}_1$-sentence A is true in a model M (for short, $M \models A$) if one of the following conditions holds.

1. A is an atomic sentence $Pc_1 \ldots c_n$ and $(\imath_M(c_1), \ldots, \imath_M(c_n)) \in \imath_M(P)$.
2. A is of the form $\neg B$ and $M \not\models B$.
3. A is of the form $(B \to C)$ and $M \not\models B$ or $M \models C$.
4. A is of the form $\forall x B$ and $M' \models B(x/c)$ for every model M' that differs from M at most in the object assigned to c, where c is the alphabetically first individual constant that does not occur in B.

Exercise 2.7

State the truth conditions for $\exists x A(x)$.

We can now define entailment. Every model of Γ is a model of A.
A and B are logically equivalent iff each entails the other.

Exercise 2.8

Show that $\Gamma, A \models B$ iff $\Gamma \models A \to B$.

Exercise 2.9

Show that $\forall x(A \to B) \models \forall x A \to \forall x B$. Show that if x is not free in B then $\forall x(A \to B)$ is equivalent to $\exists x A \to B$.

Exercise 2.10

Show that substituting a subformula by an equivalent subformula results in an equivalent formula.

Exercise 2.11

Show by induction that if M and M' are two models that differ only in the object assigned to an individual constant c, and A is an $\mathfrak{L}_1$-sentence in which c does not occur, then $M \models A$ iff $M' \models A$.

Definition 2.8: Prenex Normal Form

A formula is in *prenex normal form* if all its quantifiers are at the beginning.

The idea of the proof is to demonstrate that a quantifier which occurs somewhere in the middle of a formula can always be moved one step to the left, and by sufficiently many such steps we can bring all the quantifiers as far to the left as possible, so that they do all occur in a block at the beginning.

All we need to show is that

- A → xB is equivalent to x(A → B), if x is not free in A.

- A → ¬ xB is equivalent to ¬ x(A → B), if x is not free in A.
- alpha-equivalence

Maybe show semantically?

Needed to generalize Σ and Π: Call a formula of LPA essentially Σ_n or Π_n if it can be transformed into a Σ_n or a Π_n formula, respectively, by the usual prenexing rules.

2.4 Functions, identity, and mathematical structures

For mathematical purposes, it is useful to have a version of predicate logic with function symbols and identity. Consider the statement

$$1 + 2 = 3.$$

How could we formalize this? In $\mathfrak{L}_1$, we would have to use a three-place predicate symbol S and write

$$S(1, 2, 3).$$

But this obscures the real structure of the statement, which states an identity between 1 + 2 and 3.

The first step to remedy this situation is introduce a predicate for identity. We'll use '=' for this predicate. (We also use '=' for identity in our metalanguage; you'll have to figure out from context which is in play.) Officially, all predicates are placed in front of their arguments, so we should write '$= ab$'. In practice, we'll "abbreviate" this as '$a = b$'. We'll also write '$a \neq b$' for '$\neg = ab$'.

Exercise 2.12

Construct a sentence with '=' as the only predicate symbol that is true in a model M only if the domain of M has (a) at least two members, (b) at most two members, (c) exactly two members.

Of course, nothing stopped us from having a predicate for identity before. The real novelty is that we'll classify '=' as a *logical* expression, like '$\wedge$' and '$\forall$' and unlike 'a'

and 'F'. This means that we'll have special rules for reasoning with '=', and that models are not allowed to interpret '=' as meaning anything other than identity.

Before we get to this, I want to introduce the second addition to $\mathfrak{L}_1$ that will allow us to form complex singular terms like '1 + 2'.

The expression '$1+2$' denotes a number: the number 3. (That's why '$1+2 = 3$' is true.) How does this come about? We can understand the '+' sign as expressing a function that maps a pair of numbers to a number: 1 and 2 to 3, 2 and 5 to 7, and so on. So understood, '+' is a *function symbol*. Such symbols are ubiquitous in maths, and we'll add them to $\mathfrak{L}_1$.

First, however, a few general words on the concept of a function, which will play an important role throughout these notes.

A function, in the mathematical and logical sense, takes one or more objects of a certain kind as input and (typically) returns an object of a certain kind as output. An input to a function is also called an *argument* to the function; the output is called the function's *value* for that argument.

The addition function takes two numbers as input and returns a number. The square function takes a single number and returns a number (the square of the input). The inputs and outputs don't need to be numbers. There is an "area" function that takes a country as input and returns its size in (say) square kilometres. And there is a "mother" function that takes a person as input and returns their mother.

The class of inputs to a function is called *domain*; the class of candidate outputs are its *codomain*. If a function has domain X and codomain Y, we say that it is a function *from X to Y*. If all inputs and outputs of a function belong to a set X, we say that it is a function *on X*.

Some functions are not defined for all objects in their domain. The division function, for example, takes two numbers as input and returns a number, but it is undefined if the second input number is zero. Such functions are called *partial*.

Functions are often associated with a recipe or algorithm for determining the output for a given input. (There are algorithms for computing sums or squares.) But we don't build this into the concept of a function. Any mapping from inputs to outputs is a function, even if there is no recipe for determining the output.

Since functions are just mappings from inputs to outputs, they are fully determined by their values for each input. Consider, for example, the function g from numbers to numbers that takes a number x and as input and returns x^2 if Goldbach's conjecture is true and 0 if Goldbach's conjecture is false. Goldbach's conjecture says that every even number greater than 2 is the sum of two primes. It is not known whether the conjecture is true. So we don't know what g returns for inputs other than 0. But we know that g is either identical to the square function or to the constant function that returns 0 for every input.

Exercise 2.13

Give another example of a function with 1, 2, 3 arguments.

Now we'll add function symbols and an identity predicate to our first-order language $\mathfrak{L}_1$. Let's call the resulting language $\mathfrak{L}_1^=$. Function symbols combine with singular terms to form new singular terms. For example, if f is a two-place function symbol and a, b are individual constants, then $f(a, b)$ is a singular term; it denotes the value of the function f for the arguments a and b.

Definition 2.9

A *(singular) term* of $\mathfrak{L}_1^=$ is a finite string constructed by the following rules.

- Every variable and individual constant is a singular term.
- If f is an n-ary function symbol ($n > 0$) and $t_1, \dots, t_n$ are singular terms then $f(t_1, \dots, t_n)$ is a singular term.

By '$f(t_1, \dots, t_n)$' I mean the string that begins with f, followed by the opening parenthesis, followed by the terms $t_1, \dots, t_n$ separated by commas, and ending with a closing parenthesis.

We could allow for zero-place function symbols. A (total) zero-ary function takes nothing as input and returns a single value; a zero-place function symbol f simply denotes this value; it behaves just like a name.

A 0-ary function symbol functions syntactically and semantically just like an individual constant.

Officially, function symbols are placed in front of their arguments, with parentheses and commas to separate the arguments. $(x \times y) + z$ would be written $+(\times(x, y), z)$. For the sake of readability, we'll allow the more familiar infix notation as a metalinguistic "abbreviations".

Exercise 2.14

Consider a language of arithmetic with constants 0,1,2,3..., and + and × intended to mean addition and multiplication. Can you write the statement of Lagrange's Theorem, which states that every natural number is the sum of four squares?

Exercise 2.15

Suppose we used infix notation without parenthesis, writing, for example, x+y for the sum of x and y. This would cause a problem with more complex terms that contain functional terms. Can you explain the problem? [Think of subtraction: x-y-z has two readings. We want *unique readability*.]

The definition of formulas is the same as above.

Definition 2.10

A *formula* of $\mathfrak{L}_1^=$ is a finite string built up according to the following formation rules:

(i) If P is an n-ary predicate symbol and $t_1, \dots, t_n$ are singular terms then $Pt_1 \dots t_n$ is a formula.
(ii) If A is a formula, then so is $\neg A$.
(iii) If A and B are formulas, then so is $(A \to B)$.
(iv) If x is a variable and A is a formula then $\forall xA$ is a formula.

The axiomatic calculus can stay the same, except that we now allow for closed terms wherever we previously appealed to individual constants. That is, we have

(A4) $\forall xA \to A(x/t)$
(Gen) From $A(x/t)$ infer $\forall xA$.

We'll add some new axiom schemas for identity: for any closed terms t_1 and t_2 and formulas A in which only x is free, (A6) $t_1 = t_1$

(A7) $t_1 = t_2 \to (A(x/t_1) \to A(x/t_2))$

The somewhat strange formulation of LL is a trick that allows replacing only some occurrences of t_1 by t_2.

Let's think about models.

Singular terms don't make a claim; they merely serve to pick out some object.

We define $[t]^M$ so that $[f(t_1, \dots, t_n)]^M = [f]^M([t_1]^M, \dots, [t_n]^M)$.

In the model definition, we require function symbols to denote total functions on the domain of discourse D. Otherwise we'll have empty terms.

Definition 2.11

A *model* M of a first-order language $\mathfrak{L}_1^=$ consists of

(i) a non-empty set D, called the *domain* or *universe* of M, and
(ii) a function ι that assigns
  - to each individual constant of $\mathfrak{L}_1$ an object in D,
  - to each n-ary function symbol of $\mathfrak{L}_1^=$ an n-ary total function on D, and
  - to each non-logical n-ary predicate of $\mathfrak{L}_1$ a set of n-tuples of objects in D.

Definition 2.12

An $\mathfrak{L}_1^=$-sentence A is true in a model M (for short, $M \models A$) if one of the following conditions holds.

1. A is an identity sentence $t_1 = t_2$ and $[t_1]^M = [t_2]^M$.
2. A is any other atomic sentence $Pt_1 \dots t_n$ and $([t_1]^M, \dots,) \in \iota_M(P)$.
3. A is of the form $\neg B$ and $M \not\models B$.
4. A is of the form $(B \to C)$ and $M \not\models B$ or $M \models C$.
5. A is of the form $\forall x B$ and $M' \models B(x/c)$ for every model M' that differs from M at most in the object assigned to c, where c is the alphabetically first individual constant that does not occur in B.

Exercise 2.16

Explain why $\models a = a$, if a is an individual constant.

Exercise 2.17

Define a model in which 1+1=2 is true and one in which it is false.

Once we have function symbols, the assumption that singular terms of our formal language must not be empty becomes even more problematic than it was before. As I said, in ordinary maths, we have expressions like $1/0$ that don't denote anything. We could allow for empty terms in our models. This would have the advantage that we could then also allow for empty domains. But we'd have to change our calculus. For example, we shouldn't allow the inference from $\forall xFx$ to Fa to $\exists xFx$. There are different ways of pursuing this approach. They are studied in a non-classical predicate logic called *free logic*. For most applications we prefer to keep things simple.

Many important mathematical structures consist of a set of objects together with some operations and relations on these objects, and sometimes with a designated object playing a special role. For example, we might construe the structure of natural numbers as consisting of the set $\mathbb{N}$ of numbers 0,1,2,..., the designated object 0, the addition operation $+$, the multiplication operation $\times$, and the less-than relation $<$. It is common to package all this into a list: $(\mathbb{N}, 0, +, \times, <)$.

When we talk about this structure in a first-order language, we'll naturally use $\mathbb{N}$ as the intended domain of discourse, we'll have an individual constant for 0, function symbols intended to express $+$ and $\times$, and a predicate symbol for $<$. In effect, the structure $(\mathbb{N}, 0, +, \times, <)$ then provides a model for our language. That's why textbooks on mathematical logic often identify models with mathematical structures.

Careful: when specifying the algebra, we don't use the first-order object language. So '+' here is metalanguage. The object language may also have a '+' symbol or a '0' symbol, but they are not treated as logical. So their meaning isn't fixed. When we talk about logical entailment, we need to consider any possible way of interpreting '0' and '+'.

E.g, D = { Paris, Rome, Canberra }, +(x,y) = Paris, ×(x,y) = Rome, 0 = Canberra.

Note that, technically, we don't allow for languages with only one or two individual constants. We need an unending supply of constants for two reasons. One, the interpretation of $\forall xA$ appeals to $A(x/c)$, where c is a constant that doesn't occur in A. The interpretation of $\forall x\forall yA$ therefore involves two constants, and so on. But it doesn't matter what M assigns to these further constants. This is also true for the other reason why we need extra constants: in our calculus, we often need to reason from $\forall x A(x)$ to $A(c)$, where c is a new constant, so that if we have derived $B(c)$ from $A(c)$, we can later apply Gen to get $\forall x B(x)$. Here, too, the extra constants only serve a Hilfsfunktion. They aren't used to pick out a particular object. (Such constants are historically called *eigenvariables*.)

Definition 2.13: Isomorphic Structures

Structures $\mathscr{A}$ and $\mathscr{B}$ are *isomorphic* if there is a bijection f from the domain of $\mathscr{A}$ to the domain of $\mathscr{B}$ such that xxx

Clearly, truth at a structure is preserved under isomorphisms. A full proof is in BBJ pp.140ff. Might move to next chapter?

Exercise 2.18

Show that isomorphisms are equivalence relations

2.5 Soundness

We want to show that if $\Gamma \vdash A$, then $\Gamma \models A$. We first show that if $\vdash A$, then $modelsA$. To this end, we need two simple lemmas.

Lemma 2.1

If $M \models t_1 = t_2$ and $M \models A(x/t_1)$, then $M \models A(x/t_2)$.

Induction on the complexity of A. The base case is the semantic clause (1); the inductive steps are immediate for $\neg$, $\to$ and $\forall$.

Lemma 2.2

Let M and M' be models that differ at most in the object assigned to a constant c, and let A be an $\mathscr{L}_1$–sentence in which c does not occur. Then $M \models A$ iff $M' \models A$.

Straight induction on the complexity of A; clause (1) is used in the base case.

Soundness of the first–order calculus

If $\vdash A$ then $\models A$.

Let $B_1, \dots, B_n$ be a proof of A. We show by induction on i that

$$(*) \quad \models B_i, \; 1 \leq i \leq n.$$

The result for $i = n$ yields $\models A$.

We go through all possible cases of B_i.

(i) *B_i is an instance of A1–A3.* These are propositional tautologies; the semantic clauses for $\neg$ and $\to$ coincide with the classical truth tables, so they are true in every model.

(ii) *B_i is A4: $\forall x\, B \to B(x/t)$.* Let M be any model for which $M \models \forall x\, B$. Clause (5) of the semantics supplies a constant c not occurring in B such that for *every* model M' that differs from M only by the object interpreting c, we have $M' \models B(x/c)$. Choose $a = [t]^M$ and form M' by giving c the value a while leaving everything else as in M. Then $M' \models B(x/c)$; since c and t denote the same object in M', Lemma **??** gives $M' \models B(x/t)$. Because c is absent from $B(x/t)$, Lemma **??** transfers truth back to M: $M \models B(x/t)$. Thus every model that satisfies the antecedent satisfies the consequent; the instance is valid.

(iii) *B_i is A5: $\forall x(A \to B) \to (A \to \forall x\, B)$, $x \notin \mathsf{FV}(A)$.* Let $M \models \forall x(A \to B)$ and $M \models A$. For an arbitrary constant c and an arbitrary M' differing from M only in the value of c, clause (5) gives $M' \models (A \to B)(x/c)$, i.e. $M' \models A \to B(x/c)$. As $c \notin A$, Lemma **??** yields $M' \models A$, hence $M' \models B(x/c)$. Because c and M' were arbitrary, $M \models \forall x\, B$. Therefore the axiom instance is valid.

(iv) *B_i is A6: $t_1 = t_2$.* Every closed term has a denotation, and any object equals itself, so $M \models t_1 = t_2$ for every model M.

(v) *B_i is A7: $t_1 = t_2 \to (B(x/t_1) \to B(x/t_2))$.* Suppose $M \models t_1 = t_2$ and $M \models B(x/t_1)$. Then $[t_1]^M = [t_2]^M$, and Lemma **??** gives $M \models B(x/t_2)$. Hence the conditional is true in all models.

(vi) *B_i is obtained by MP from earlier lines C and $C \to B_i$.* By the induction hypothesis C and $C \to B_i$ are valid; in every model clause (4) for $\to$ forces B_i to be true.

(vii) *B_i is obtained from an earlier line B by Gen.*

Fix an arbitrary model M and assume $M \models B$ (true by the induction hypothesis). To prove $M \models \forall x\, B(t/x)$ we must show: for *every* constant c not in B and *every* model M' differing from M only in the value of c, we have $M' \models B(x/c)$.

Choose such c and M'. Build a model M'' that agrees with M' except that it interprets t as the object that M' assigns to c. Because $t \notin B$ (by (G)), M'' and M' interpret every symbol of $B(x/c)$ alike; by Lemma **??** it suffices to verify $M'' \models B(x/c)$. But in M'' the two closed terms c and t denote the same element, and $M'' \models B(x/t)$ (truth is preserved when we changed only the value of t), so Lemma **??** gives $M'' \models B(x/c)$, completing the requirement. Clause (5) now yields $M \models \forall x\, B$.

This shows that every axiom is valid and that MP and Gen preserve validity.

Corollary 2.1: I

$\Gamma \vdash A$, then $\Gamma \models A$.

Assume there is a deduction of $B_1, \dots, B_n$ of A from Γ. This deduction can involve only finitely many sentences $A_1, \dots, A_m$ in Γ. So we also have

$$A_1, \dots, A_m \vdash A.$$

By the deduction theorem, it follows that

$$\vdash A_1 \to (A_2 \to (\dots (A_m \to A) \dots)).$$

By the soundness theorem, it implies that

$$\models A_1 \to (A_2 \to (\dots (A_m \to A) \dots)).$$

But if $\Gamma \models A \to B$ then $\Gamma, A \models B$ [exercise!]. So we have

$$A_1, \dots, A_m \models A.$$

Since $\{A_1, \dots, A_m\} \subseteq \Gamma$, it follows that $\Gamma \models A$.

Exercise 2.19

Can we still use soundness to prove consistency of the axiomatization? Yes.

3 Completeness

In this chapter, we meet some important metalogical results. The first is the completeness theorem. This shows that different approaches to first-order logical consequence coincide. It shows that there is a mechanical way to check for first-order entailment. This is a great strength of first-order logic (not shared by second-order logic). But it is tightly connected to some limitations: compactness and Löwenheim-Skolem.

3.1 Plan of the completeness proof

We'll prove completeness. This will make it plausible that all mathematical reasoning can indeed be the captured in our first-order calculus. Almost all mathematical statements can be expressed in a suitable first-order language. Before Gödel's completeness proof, it was known that a lot of ordinary mathematical inferences could be replicated in the first-order calculus that we've reviewed in the previous chapter, but it was an open question whether this is true for all such inferences. The completeness proof provides strong evidence for a positive answer. It shows that any form of inference that *can't* be replicated in the first-order calculus can lead from true premises to false conclusions.

Completeness is surprising. Think about the usefulness of the semantic conception of entailment. Any model where this is true is a model where that is true. No need to posit a finite derivation. Model-theoretic validity is a highly infinitary notion. Why should it map onto finitary proofs? (We seemingly need to check every possible model.)

More dramatically: suppose I like the number 1, the number 2, the number 3, etc. It follows model-theoretically that I like all numbers. But proofs are finite. So you can't prove the conclusion from the premises. Yet we have completeness for FOL!

What about the numbers argument? This argument isn't *logically* valid. It's *arithmetically* valid, we might say: valid given the standard meaning of the arithmetical terms. Arithmetical entailment is not mirrored by any finite proof system!

Completeness entails that all (first-order) logical truths are provable. Obviously, not all truths are logical truths. So completeness doesn't mean that all truths are provable.

Exercise 3.1

Which, if any, of these statements are correct? (1) The soundness theorem implies that every sentence provable from the Peano axioms is a true sentence of arithmetic; (2) The completeness theorem implies that every true sentence of arithmetic is provable from the Peano axioms. [yes/no]

We'll use not Gödel's 1929 technique, but Henkin's. We've already used this technique in ch.1 as a warm up. Let's recall the key ideas.

The proof is by contraposition. We assume $\Gamma \nvdash A$ and show that, in any such case, $\Gamma \nvDash A$.

1. Assume $\Gamma \nvdash A$.
2. Infer that $\Gamma \cup \{\neg A\}$ is consistent.
3. Show that $\Gamma \cup \{\neg A\}$ can be extended to a maximal consistent set Γ^+.
4. Construct a model M based on Γ^+ in which all members of Γ^+ are true.
5. Infer that $\Gamma \nvDash A$.

Steps 2 and 5 are easy, and proceed just like in chapter 1. The following lemma established step 2.

Lemma 3.1

If $\Gamma \nvdash A$ then $\Gamma \cup \neg A$ is consistent.

Assume the contrary. Then $\Gamma \cup \neg A \vdash B$ and $\Gamma \cup \neg A \vdash \neg B$ for some closed sentence B. By the deduction theorem, $\Gamma \vdash \neg A \to B$ and $\Gamma \vdash \neg A \to \neg B$. So $\Gamma \vdash \neg\neg A$ (by *Ex Falso Quodlibet*), and therefore $\Gamma \vdash A$ (by double negation elimination). Contradiction.

TODO: Explain step 5?

It remains to fill in steps 2 and 3. In effect, what remains to show is this:

Every consistent set of $\mathfrak{L}$-sentences has a model.

Let's focus on this. We'll show how, given any consistent set Γ of sentences in a first-order language $\mathfrak{L}$, we can construct a model M_Γ in which all sentences in Γ are true.

As in chapter 1, it helps to extend Γ to a maximal consistent set Γ^+. If, say, Γ contains $Fa \lor Gb$, it's not obvious if we should have build model in which Fa is true, or Gb is true, or both. Γ^+ will contain Fa or Gb or both, so it will answer this question. Clearly, if our model is a model of Γ^+, then it will be a model of Γ.

Now suppose we have Fa in Γ^+. Our model M must then contain an object $[a]^M$ denoted by a that falls in the extension $[F]^M$ of F. The nature of that object is irrelevant. For convenience, we'll choose *the individual constant a* as the object denoted by a. Then we can say that the extension of F comprises all individual constants c for which $Fc \in \Gamma^+$.

TODO: say a little more about this. It's odd to have a name for a name. But that's allowed. Why it helps.

But there's a problem. Suppose Γ contains $\exists x Fx$. We want this to be true in M. So the extension of F in M must be non-empty. If, as I just said, the extension of F comprises all individual constants c for which $Fc \in \Gamma^+$, this means that there must be some constant c such that $Fc \in \Gamma^+$. (Otherwise $\exists x Fx$ would not be true in M.) So we need to ensure that there is such a "witnessing" sentence Fc in Γ^+, for each existential sentence $\exists x Fx$ in Γ^+. Now the problem is that Γ may already contain $\neg Fc$ for every individual constant c, and then we can't add the required witness!

To get around this problem, we'll let Γ^+ be a set in an extend ed language $\mathfrak{L}^+$ that adds new individual constants to $\mathfrak{L}$. We can then use these constants to ensure that every existential sentence in Γ has a witness.

There is one more complication, arising from the fact that we have the identity symbol in $\mathfrak{L}$ (and $\mathfrak{L}^+$). If we let each individual constant denote itself, any identity statement involving different individual constants will be false. After all, no constant is identical to any other constant. But $a = b$ is consistent, and might occur in Γ and therefore in Γ^+. So, really, we let each constant denote a *set* of terms. The constant a will denote the set $\{c \mid a = t \in \Gamma^+\}$.

3.2 The proof

Let Γ be a consistent set of $\mathfrak{L}$-sentences. Let's remind ourself of what this means.

Definition 3.1

A set Γ of sentences in a first-order language $\mathfrak{L}$ is *consistent* (within $\mathfrak{L}$) if there is no $\mathfrak{L}$-sentence A in the language such that $\Gamma \vdash A$ and $\Gamma \vdash \neg A$.

We'll extend Γ to a maximal consistent set in a language $\mathfrak{L}^+$ with infinitely many new individual constants. We first confirm that Γ is consistent within the extended language $\mathfrak{L}^+$. This is not entirely trivial because there are axioms in $\mathfrak{L}^+$ that aren't in $\mathfrak{L}$. We need to confirm that these new axioms don't allow deriving a contradiction from Γ.

Lemma 3.2

If Γ is a set of $\mathfrak{L}$-sentences that is consistent within $\mathfrak{L}$, and $\mathfrak{L}^+$ extends $\mathfrak{L}$ by a set of new individual constants, then Γ is consistent within $\mathfrak{L}^+$.

Proof. Assume for contraposition that Γ is inconsistent within $\mathfrak{L}^+$. Then there is a deduction $A_1, \dots, A_n$ of $\bot$ from Γ and the axioms in $\mathfrak{L}^+$. Being finite, this deduction only uses finitely many of the new constants in $\mathfrak{L}^+$. Call them $c_1, \dots, c_k$. The deduction also uses only finitely many of the old constants in $\mathfrak{L}$. Since $\mathfrak{L}$ has infinitely many constants, we can choose k distinct constants $d_1, \dots, d_k$ from $\mathfrak{L}$ that don't occur in the deduction. Consider the sequence of sentences $A'_1, \dots, A'_n$ that results from $A_1, \dots, A_n$ by replacing each new c_i by d_i. It is easy to see that

(i) If A_i is an axiom then so is A'_i;

(ii) If $A_i \in \Gamma$ then $A'_i \in \Gamma$ (because sentences in Γ contain no new constants);

(iii) If A_i follows from $A_1, \dots, A_{i-1}$ by MP or Gen, then A'_i follows from $A'_1, \dots, A'_{i-1}$ by MP or Gen.

So $A'_1, \dots, A'_n$ is a deduction of $\bot$ from Γ and the axioms in $\mathfrak{L}$, meaning that Γ is inconsistent within $\mathfrak{L}$.

Now we want to show that we can extend Γ to a maximal consistent set Γ^+ in which every sentence $\neg\forall x A$ has a witness $\neg A(x/c)$.

Definition 3.2

A set of sentences Γ in a first-order language $\mathfrak{L}$ is *maximal consistent* within $\mathfrak{L}$ if it is consistent and for every $\mathfrak{L}$-sentence A, either $A \in \Gamma$ or $\neg A \in \Gamma$.

Definition 3.3

A set of sentences Γ in a first-order language $\mathfrak{L}$ is a *Henkin set in* $\mathfrak{L}$ if it is maximal consistent and for every $\mathfrak{L}$-formula A in which a single variable x is free, if $\neg\forall x A$ is in Γ then there is an individual constant c for which $\neg A(x/c)$ is in Γ.

The following lemma will be useful.

Lemma 3.3

If Γ is maximal consistent and $\Gamma \vdash A$ then $A \in \Gamma$.

[exactly as in the propositional case?] Assume Γ is maximal consistent and $\Gamma \vdash A$. If $A \notin \Gamma$ then $\neg A \in \Gamma$ by maximality, and $\Gamma \vdash \neg A$ by the Identity property of $\vdash$. This would render Γ inconsistent. So $A \in \Gamma$.

The next lemma is our first major step.

Lemma 3.4

Every consistent set of $\mathfrak{L}$-sentences Γ can be extended to a Henkin set in any language $\mathfrak{L}^+$ that adds infinitely many individual constants to $\mathfrak{L}$.

Let Γ be a consistent set of $\mathfrak{L}$-sentences. Let $A_1, A_2, \ldots$ be a list of all $\mathfrak{L}^+$-formulas with exactly one free variable. We define a sequence of sets $\Gamma_0, \Gamma_1, \ldots$ as follows:

$$\begin{aligned} \Gamma_0 &:= \Gamma \\ \Gamma_{n+1} &:= \Gamma_n \cup \{\neg\forall x A_n \to \neg A_n(x/c_n)\}, \end{aligned}$$

where x is the free variable in A_n and c_n is a new $\mathfrak{L}^+$-constant that does not occur in Γ_n. (There must be some such constant because $\mathfrak{L}^+$ contains infinitely many constants that don't occur in Γ.) Let Γ' be the union $\bigcup_n \Gamma_n$ of all sets in this sequence. (That is, a sentence A is in Γ' iff it is in some Γ_n.)

We show that Γ' is consistent. Suppose not. Then there is a derivation of $\bot$ from Γ and the "Henkin sentences" $\neg\forall x A_n \to \neg A_n(x/c_n)$. This derivation can use only finitely many of the Henkin sentences. So one of the Γ_n must be inconsistent. But we can show by induction on n that each Γ_n is consistent.

The base case, for $n = 0$, hold by assumption: Γ is consistent.

For the inductive step, assume Γ_n is consistent and suppose for reductio that Γ_{n+1} is inconsistent. Then $\Gamma_n \vdash \neg(\neg\forall x A_n \to \neg A_n(x/c_n))$ by lemma xxx. And then by lemma xxx and DNE,

(i) $\Gamma_n \vdash \neg\forall x A_n$, and

(ii) $\Gamma_n \vdash \neg\neg A_n(x/c_n)$.

From (ii), we get $\Gamma_n \vdash A_n(x/c_n)$ by DNE. As c_n does not occur in Γ_n, we have

(iii) $\Gamma_n \vdash \forall x A_n$ by UG.

(i) and (iii) contradict the assumption that Γ_n is consistent.

Next, we need to extend Γ' to a maximal consistent set Γ^+. The construction follows the proof of Lindenbaum's Lemma (xxx). Let $S_1, S_2, \ldots$ be a list of all $\mathfrak{L}^+$-sentences. Starting with Γ', We define another sequence of sets $\Gamma'_0, \Gamma'_1, \ldots$:

$$\Gamma'_0 := \Gamma'$$
$$\Gamma'_{n+1} := \begin{cases} \Gamma'_n \cup \{S_n\} & \text{if } \Gamma'_n \cup \{S_n\} \text{ is consistent,} \\ \Gamma'_n \cup \{\neg S_n\} & \text{otherwise.} \end{cases}$$

Let Γ^+ be the union $\bigcup_n \Gamma'_n$ of all sets in this sequence.

The maximal consistency of Γ^+ follows exactly as in the proof of Lindenbaum's Lemma. It remains to show that Γ^+ has the witnessing property, meaning that for every sentence $\neg\forall x A$ in Γ^+, there is a corresponding sentence $\neg A(x/c)$ in Γ^+.

Let A be any formula in which x is the only free variable. By construction, Γ' contains $\neg\forall x A \to \neg A(x/c)$, for some constant c. Since Γ^+ extends Γ', it also contains this sentence. If Γ^+ contains $\neg\forall x A$, it must contain $\neg A(x/c)$ as well, as otherwise it would contain $A(x/c)$, which would make it inconsistent.

Done!

Next, we show how to read off a model M_{Γ^+} from a Henkin set Γ^+. The domain D of M_{Γ^+} will consist of sets of $\mathfrak{L}^+$-terms. We could simply say that

$$D = \{\{s \mid t = s \in \Gamma^+\} \mid t \text{ is a closed } \mathfrak{L}^+ \text{ term}\}.$$

But let's be more explicit about what this amounts to.

Definition 3.4

A binary relation R on some domain D is an *equivalence relation* if it is

(i) *reflexive*: for every $x \in D$, xRx;
(ii) *symmetric*: for every $x, y \in D$, if xRy then yRx; and
(iii) *transitive*: for every $x, y, z \in D$, if xRy and yRz then xRz.

Lemma 3.5

If Γ is a Henkin set then the relation $\sim_\Gamma$ that holds between $\mathfrak{L}^+$-terms t, s iff $t = s \in \Gamma$ is an equivalence relation.

check!

- Reflexivity: by axiom A6, $\Gamma^+ \vdash^+ t = t$; so $t = t \in \Gamma^+$.
- Symmetry: $\Gamma^+ \vdash^+ t = s \to s = t$, from A7 with A the formula $x = y$). So if $t = s \in \Gamma^+$ then $s = t \in \Gamma^+$.
- Transitivity: $\Gamma^+ \vdash t = s \to (s = u \to t = u)$, by xxx. So if $t = s \in \Gamma^+$ and $s = u \in \Gamma^+$ then $t = u \in \Gamma^+$.

An equivalence relation *partitions* the domain over which it is defined into distinct cells, within which all objects stand in the relation to one another. These cells are called *equivalence classes*. If R is an equivalence relation and x an object in the domain, we write '$[x]_R$' for the equivalence class (of R) that contains x. That is, $[x]_R = \{y \in D \mid xRy\}$.

We can now specify the domain of M_{Γ^+} as follows:

$$D = \{[t]_\sim \mid t \text{ is a closed } \mathfrak{L}^+\text{-term}\},$$

where $\sim$ is $\sim_{\Gamma^+}$.

Each individual constant c will denote $[c]_\sim$. We'll extend this idea to function terms: $f(c)$ will denote $[f(c)]_\sim$. But we can't directly assign a denotation to $f(c)$. Instead, we have to interpret f as denoting a function on D. By definition 1.5, the denotation of $f(c)$ is the denotation of f applies to that of c. Since the denotation of c is $[c]_\sim$, we want the denotation of f to be a function that returns $[f(c)]_\sim$ for input $[c]_\sim$. So we'll stipulate that for any one-place function symbol f and closed term t,

$$[f]^M([t]_\sim) = [f(t)]_\sim.$$

This kind of stipulation can go wrong. Suppose $[t]_\sim$ contains two terms s and t, and $[f(s]_\sim \neq [f(t)]_\sim$. Then what is the value of $[f]^M$ for $[t]_\sim$? Our stipulation appears to say that

$$[f]^M([s]_\sim) = [f(s]_\sim)$$

and

$$[f]^M([t]_\sim) = [f(t]_\sim),$$

But $[s]_\sim$ is the same object as $[t]_\sim$, and a function must return the same value for the same input. To legitimize our stipulation, we must therefore show that this problem can never arise.

A similar issue arises when we define the interpretation of predicates. To ensure that Ft is in Γ iff it is true in M, we'll stipulate that

$$[t]_\sim \in [F]^M \text{ iff } Ft \in \Gamma.$$

Here, too, we have to ensure that if s and t are both in $[t]_\sim$, we can never have only one of Fs and Fs in Γ.

The following lemma shows that neither problem can arise, legitimizing the following definition.

Lemma 3.6

If f is an n-ary function symbol, P an n-ary predicate symbol, and $s_1, \dots, s_n, t_1, \dots, t_n$ are terms such that $[s_1]_\sim = [t_1]_\sim, \dots, [s_n]_\sim = [t_n]_\sim$, then

(i) $[f(s_1, \dots, s_n)]_\sim = [f(t_1, \dots, t_n)]_\sim$;

(ii) $P(s_1, \dots, s_n) \in \Gamma \quad \text{iff} \quad P(t_1, \dots, t_n) \in \Gamma$.

Assume $[s_i]_\sim = [t_i]_\sim$, for $i = 1, 2, \dots, n$. Then $s_i = t_i$ is in Γ, for each such i.

(i). By axiom A6, $f(s_1, \dots, s_n) = f(s_1, \dots, s_n)$ is in Γ. By n instances of A7 and MP, it follows that $f(s_1, \dots, s_n) = f(t_1, \dots, t_n)$ is in Γ, which entails that $[f(s_1, \dots, s_n)]_\sim =$

$[f(t_1, \dots, t_n)]_\sim$.

(ii). Assume $P(s_1, \dots, s_n) \in \Gamma$. By n instances of A7 and MP, $P(t_1, \dots, t_n)$ is in Γ as well. The converse holds by the same reasoning.

Definition 3.5

The *Henkin model* M_Γ for a Henkin set Γ in a language $\mathfrak{L}$ is defined as follows. The domain D of M_Γ is the set $\{[t]_\sim \mid t \text{ is a closed } \mathfrak{L}\text{-term}\}$, where $\sim$ is the equivalence relation $\sim_\Gamma$.
The interpretation function of M is defined as:

(i) If c is an individual constant then $c^M = [c]_\sim$.

(ii) If f is an n-ary function symbol then $f^M([t_1]_\sim, \dots, [t_n]_\sim) = [f(t_1, \dots, t_n)]_\sim$.

(iii) If P is an n-ary predicate symbol then $([t_1]_\sim, \dots, [t_n]_\sim) \in R^M$ iff $R(t_1, \dots, t_n) \in \Gamma$

Now we're almost ready to prove our next big lemma: that a sentence is true in the Henkin model M_Γ iff it is in the Henkin set Γ. We'll only need the following small lemma about terms.

Lemma 3.7

$t^M = [t]_\sim$ for every closed term t.

The proof is by induction on the complexity of t. The base case is covered by clause (i) in definition 3.5. So let t be $f(t_1, \dots, t_n)$. Then

$$\begin{aligned}
[f(t_1, \dots, t_n)]^M &= [f]^M([t_1]^M, \dots, [t_n]^M) && \text{by def. \textbf{??}}\\
&= [f]^M([t_1]_\sim, \dots, [t_n]_\sim) && \text{by ind. hyp.}\\
&= [f(t_1, \dots, t_n)]_\sim && \text{by def. 3.5.}
\end{aligned}$$

Here's the big lemma.

Lemma 3.8: Truth Lemma

For every closed sentence A,

$$M_\Gamma \models A \quad \text{iff} \quad A \in \Gamma.$$

By induction on A.

(i) A is an atomic sentence $P(t_1, \dots, t_n)$, where P is non-logical. $M_\Gamma \models P(t_1, \dots, t_n)$ iff $([t_1]^M, \dots, [t_n]^M) \in [P]^M$, by definition **??**, iff $([t_1]_\sim, \dots, [t_n]_\sim) \in [P]^M$, by lemma 3.7, iff $P(t_1, \dots, t_n) \in \Gamma$, by definition 3.5.

(ii) A is an identity sentence $s = t$. $M_\Gamma \models s = t$ iff $[s]^M = [t]^M$, by definition **??**, iff $[s]_\sim = [t]_\sim$, by lemma 3.7, iff $s = t \in \Gamma$, by xxx.

(iii) A is a $\neg B$. We first show that if $M_\Gamma \models \neg B$ then $\neg B \in \Gamma$. Assume $M_\Gamma \models \neg B$. Then $M_\Gamma \not\models B$ by definition **??**, so $B \notin \Gamma$ by induction hypothesis. By maximality, we must have $\neg B \in \Gamma$.

Next we show that if $\neg B \in \Gamma$ then $M_\Gamma \models \neg B$. Assume $\neg B \in \Gamma$. Then $B \notin \Gamma$ because Γ is consistent. So $M_\Gamma \not\models B$, by induction hypothesis.

(iv) A is $B \to C$. Again we take the two directions in turn. Assume $M_\Gamma \models B \to C$. Then $M_\Gamma \not\models B$ or $M_\Gamma \models C$ by definition **??**. If $M_\Gamma \not\models B$, then by induction hypothesis $B \notin \Gamma$, so by maximality $\neg B \in \Gamma$. Since $\neg B \vdash B \to C$, it follows that $B \to C \in \Gamma$. If $M_\Gamma \models C$, then by induction hypothesis $C \in \Gamma$. Since $C \vdash B \to C$, it again follows that $B \to C \in \Gamma$.

For the converse direction, assume $B \to C \in \Gamma$. We consider two cases, depending on whether $B \in \Gamma$. If $B \in \Gamma$, then $C \in \Gamma$ by MP, and by induction hypothesis $M_\Gamma \models C$. By definition **??**, this means that $M_\Gamma \models B \to C$. If, on the other hand, $B \notin \Gamma$ then by induction hypothesis $M_\Gamma \not\models B$, and $M_\Gamma \models B \to C$ by definition **??**.

(v) A is $\forall x B$. We first show that $M_\Gamma \models \forall x B$ iff $M_\Gamma \models B(x/t)$ for every closed term t. By definition **??**, $M_\Gamma \models \forall x B$ iff $M' \models B(x/c)$ for every model M' that differs from M_Γ at most in the object assigned to c, where c is a constant that does not occur in B. Since M' has the same domain as M, any object assigned to c in M' must be a set of closed terms. That is, there must be a term t such that $[c]^{M'} = [t]^M$. By the transportation theorem [???], $B(x/c)$ is true in M' iff

$B(x/t)$ is true in M [because the two sentences only differ wrt the choice of a name, and the two models coincide wrt the interpretation of the chosen name]. So $M_\Gamma \models \forall x B$ iff $M_\Gamma \models B(x/t)$ for every closed term t.

Now we can prove the two directions of the equivalence. Assume $M_\Gamma \models \forall x B$. Then $M_\Gamma \models B(x/t)$ for every closed term t, as we just showed. By induction hypothesis, $B(x/t) \in \Gamma$ for every such t. Suppose for contradiction that $\forall x B \notin \Gamma$. Then $\neg\forall x B \in \Gamma$ by maximality and by the Henkin property [xxx] there is a constant c such that $\neg B(x/c) \in \Gamma$, contradicting the fact that $B(x/t) \in \Gamma$ for every closed term t.

Conversely, assume $\forall x B \in \Gamma$. Since $\forall x B \vdash B(x/t)$ for every closed term t, we have $B(x/t) \in \Gamma$ for every closed term t by Lemma **??**. By induction hypothesis, $M_\Gamma \models B(x/t)$ for every closed term t. So $M_\Gamma \models \forall x B$.

Phew! Now we have all the ingredients we needed to show that every consistent set of sentences has a model. TODO: explain. And with that, we've essentially proved the Completeness Theorem.

Completeness Theorem (Gödel 1929)

If $\Gamma \models A$ then $\Gamma \vdash A$.

Contraposition. Assume $\Gamma \nvdash A$. Then $\Gamma \cup \{\neg A\}$ is consistent by Lemma **??**. By Lemma 3.4, we can extend Γ to a Henkin set Γ^+ in an extended language $\mathfrak{L}^+$. By the Truth Lemma, every sentence in Γ^+ is true in the Henkin model M_{Γ^+}. So in particular, xxx

[Technically, M is not a model of $\Gamma \cup \{\neg A\}$, because it interprets the new constants in $\mathfrak{L}^+$? Or is it?]

The reduct of this model to the original language satisfies T, as desired.

Exercise 3.2

Assume that $\Gamma_0, \Gamma_1, \ldots$ are consistent sets of sentences in a first-order language $\mathfrak{L}$, and that each Γ_i is a subset of Γ_{i+1}. Show that their union $\bigcup_i \Gamma_i$ is consistent.

3.3 Cardinalities

As I mentioned earlier, the completeness theorem reveals important limitations of the expressive power of first-order languages. Some of these limitations have something to do with the size of models, by which we mean the size of a model's domain. To state and appreciate these facts, let's take a break from logic and fill in some background about the sizes of sets. (This will be needed for later chapters, too.)

For finite sets, size is straightforward. The set $\{KurtGdel\}$ has size 1. The set $\{Edinburgh, Paris, Canber$ has size 3. But not all sets are finite. Consider the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. What is its size?

The official set-theoretic term for the size of a set is *cardinality*. So { Edinburgh, Paris, Canberra} has cardinality 3. Finite sets have a finite cardinality, which is simply a natural numbers. But infinite sets also have a cardinality. These aren't natural numbers, but numbers of a more general sort, called *cardinals*. The infinite cardinals are also known as the *alephs*, because they are denoted by the Hebrew letter $\aleph$ (pronounced 'aleph'). The cardinality of $\mathbb{N}$ is $\aleph_0$. Other infinite sets have cardinality $\aleph_1$, $\aleph_2$, and so on.

You may be puzzled what sort of things the alephs are. I'll give an answer in the next chapter. But note that it is equally unclear what sort of thing the natural numbers are. The more important question is how the alephs behave. What determines whether an infinite set has size $\aleph_0$ or $\aleph_1$ or whatever? That is, what's the principle for assigning cardinal numbers to sets?

The crucial move was already made by Galileo and Hume. They proposed, in effect, that two sets have the same cardinality if they can be put into a one-to-one correspondence.

To make this precise, we need the notion of a bijection.

Definition 3.6

A function $f : A \to B$ is a *bijection* if it is both injective and surjective. That is, f is a bijection if

(i) for every $x, y \in A$, if $f(x) = f(y)$ then $x = y$ (injectivity); and

(ii) for every $b \in B$, there exists some $a \in A$ such that $f(a) = b$ (surjectivity).

Intuitively, a bijection pairs up each element of A with exactly one element of B, and vice versa. Every element gets a unique partner. For finite sets, this happens precisely when both sets have the same number of elements.

Definition 3.7

Two sets A and B are *equinumerous* if there exists a bijection $f : A \to B$.

By Hume's principle, we can know say that each set that is equinumerous with $\mathbb{N}$ has cardinality $\aleph_0$. $\aleph_1$ is defined as the next larger cardinal after $\aleph_0$. This means that the smallest sets that are not equinumerous with $\mathbb{N}$ have cardinality $\aleph_1$.

Let's look at some examples. Consider the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ and the set of odd numbers $\{1, 3, 5, 7, \ldots\}$. The function $f : \mathbb{N} \to \{1, 3, 5, 7, \ldots\}$ defined by $f(n) = 2n+1$ is a bijection. It maps $0 \mapsto 1$, $1 \mapsto 3$, $2 \mapsto 5$, and so on. So according to our definition, the natural numbers and the odd numbers are equinumerous—they have the same 'size'—even though the odd numbers form a proper subset of the natural numbers. This violates the intuitive principle that the whole is greater than the part.

Exercise 3.3

Show that the set of even natural numbers $\{0, 2, 4, 6, \ldots\}$ is equinumerous with $\mathbb{N}$.

This phenomenon—that infinite sets can be equinumerous with their proper subsets—is characteristic of infinite sets. Indeed, it can be taken as the *definition* of what it means for a set to be infinite.

We now introduce some standard terminology for infinite sets.

Definition 3.8

A set A is *countable* if it is either finite or equinumerous with $\mathbb{N}$. A set that is equinumerous with $\mathbb{N}$ is called *countably infinite* or *denumerable*.

So the natural numbers are countably infinite. As we've seen, so are the odd numbers and the even numbers. The integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ are also countably infinite. We can establish a bijection between $\mathbb{N}$ and $\mathbb{Z}$ by mapping:

$$\begin{aligned}
0 &\mapsto 0\\
1 &\mapsto 1\\
2 &\mapsto -1\\
3 &\mapsto 2\\
4 &\mapsto -2\\
5 &\mapsto 3\\
&\vdots
\end{aligned}$$

In general, we map even numbers $2n$ to positive integers n, and odd numbers $2n + 1$ to negative integers $-n$.

More surprisingly, the rational numbers $\mathbb{Q}$ are also countably infinite. This can be shown using Cantor's ingenious *zig-zag method*.

The idea is to arrange all positive rational numbers in an infinite grid, then traverse this grid in a systematic zig-zag pattern to create a sequence. We arrange the fractions $\frac{p}{q}$ (in lowest terms) so that $\frac{p}{q}$ appears in row p and column q:

	1	2	3	4	5	$\cdots$
1	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\cdots$
2	$\frac{2}{1}$	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{2}{5}$	$\cdots$
3	$\frac{3}{1}$	$\frac{3}{2}$	$\frac{3}{3}$	$\frac{3}{4}$	$\frac{3}{5}$	$\cdots$
4	$\frac{4}{1}$	$\frac{4}{2}$	$\frac{4}{3}$	$\frac{4}{4}$	$\frac{4}{5}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Now we traverse this grid along diagonals, starting from the top-left: $\frac{1}{1}$, then $\frac{1}{2}, \frac{2}{1}$, then $\frac{1}{3}, \frac{2}{2}, \frac{3}{1}$, then $\frac{1}{4}, \frac{2}{3}, \frac{3}{2}, \frac{4}{1}$, and so forth. We skip any fraction that isn't in lowest terms (like $\frac{2}{2} = 1$ or $\frac{2}{4} = \frac{1}{2}$) since it's already been counted.

This gives us the sequence: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{3}{1}, \frac{1}{4}, \frac{2}{3}, \frac{3}{2}, \frac{4}{1}, \frac{1}{5}, \cdots$

Each positive rational appears exactly once in this list. To handle negative rationals and zero, we can interleave them: list zero first, then alternate between positive and negative rationals. This establishes a bijection between $\mathbb{N}$ and $\mathbb{Q}$.

Exercise 3.4

Show that the set $\mathbb{N} \times \mathbb{N} = \{(m, n) \mid m, n \in \mathbb{N}\}$ of ordered pairs of natural numbers is countably infinite.

So far, every infinite set we've encountered has been countably infinite. One might wonder whether all infinite sets have the same cardinality. Cantor showed that this is not the case using a powerful technique called *diagonalization*.

Cantor's Theorem

For any set A, the power set $\mathcal{P}(A) = \{X \mid X \subseteq A\}$ has strictly greater cardinality than A. That is, there is no bijection between A and $\mathcal{P}(A)$.

We show that no function $f : A \to \mathcal{P}(A)$ can be surjective. The key insight is to construct a subset of A that cannot be in the range of f by considering the *diagonal* elements.

Suppose $f : A \to \mathcal{P}(A)$ is any function. For each element $a \in A$, we have $f(a) \subseteq A$. Either $a \in f(a)$ or $a \notin f(a)$.

The diagonalization step: Define $D = \{a \in A \mid a \notin f(a)\}$. We call this the *diagonal set* because we're considering each element a and asking whether it belongs to $f(a)$—that is, we're looking at the 'diagonal' entries in the correspondence.

Since $D \subseteq A$, we have $D \in \mathcal{P}(A)$. We claim that D is not in the range of f.

Suppose for contradiction that $D = f(b)$ for some $b \in A$. Now we ask: is $b \in D$ or $b \notin D$?

Case 1: $b \in D$. By definition of D, this means $b \notin f(b)$. But $f(b) = D$, so $b \notin D$. Contradiction.

Case 2: $b \notin D$. Since $f(b) = D$, we have $b \notin f(b)$. By definition of D, this means $b \in D$. Contradiction.

The diagonal element b cannot consistently belong to the diagonal set D! Therefore, D cannot equal $f(b)$ for any $b \in A$. So f is not surjective.

Since this holds for any function $f : A \to \mathcal{P}(A)$, there is no surjection from A to $\mathcal{P}(A)$, and hence no bijection.

This diagonalization argument is extraordinarily important. We'll see similar reasoning later when we encounter Gödel's incompleteness theorems, where diagonalization shows that no formal system can prove its own consistency.

Applied to the natural numbers, Cantor's theorem shows that $\mathcal{P}(\mathbb{N})$—the set of all subsets of natural numbers—is uncountably infinite. Its cardinality is denoted $2^{\aleph_0}$ or $\mathfrak{c}$ (the cardinality of the continuum). In fact, $\mathcal{P}(\mathbb{N})$ is equinumerous with the real numbers $\mathbb{R}$.

Exercise 3.5

Show that if A and B are countably infinite sets, then $A \cup B$ is countably infinite.

Exercise 3.6

Show that the set of finite subsets of $\mathbb{N}$ is countably infinite.

Cantor's theorem reveals a hierarchy of infinities. Starting with any infinite set, we can always construct a strictly larger infinite set by taking its power set. From $\mathbb{N}$ we get $\mathscr{P}(\mathbb{N})$, then $\mathscr{P}(\mathscr{P}(\mathbb{N}))$, and so on, creating an endless sequence of ever-larger infinities. This shatters any simple picture of infinity as a single, uniform concept.

3.4 Unintended Models

Now return to first-order logic. Suppose we want to formalize reasoning about, say, the natural numbers, in a suitable first-order language $\mathfrak{L}$. This language might have a constant '0', function symbols '+' and '×', and so on. We can say things like

$$\forall x(x + 0 = x)$$

or

$$\forall x(x \times 0 = x).$$

Intuitively, the first sentence is true and the second false. But wait! Officially, truth and falsity are defined only relative to a model. It is easy to construct a model in which the second sentence is true.

Exercise 3.7

Do it.

Logic abstracts away from the meaning of non-logical expressions. '0', '+', '×' are non-logical. Neither of the two sentences is *logically true*, true in all models.

Still, there's a good sense in which the first sentence is true and the second false. When we use the language, we have a particular interpretation in mind. We use '0' for the number zero, '+' for addition, '×' for multiplication. We also assume that the quantifiers range over the set of natural numbers. So understood, the first sentence says that adding zero to any number yields that same number, and the second says that multiplying any number by zero yields that same number.

We'll say that *intended model* of the language $\mathfrak{L}$ is the model $\mathscr{N}$ whose domain is the set of natural numbers, and in which '0' refers to zero, '+' to addition, and '×' to multiplication.

We might hope that we can rule out non-intended models by laying down some postulates in the language. For example, we can rule out models with only one object by the following postulate:

$$\exists x \exists y (x \neq y).$$

We can similarly find a postulate that rules out models with only two objects. And so on.

there are at least two objects.

of course, there will always be non-intended models because the nature of the elements in D doesn't matter. That is, given a model, I can define a new model in which the number 7 is swapped by Caesar etc. The best we can hope for is to narrow the models down to isomorphism.

Define Isomorphism.

Define elementary equivalence.

Proof sketch: isomorphic models are elementarily equivalent.

If we had the converse – which we don't – we'd know that non-isomorphic models can always be distinguished by some sentence. So that we can always rule out unintended non-isomorphic models by laying down another postulate, although we might need infinitely many postulates, but never mind that for now. In fact, for present purposes we can consider whether it would suffice to lay down as postulate every sentence in the language that's true in the intended model.

Definition: If M is a model for a language L, the /theory of M/ is the set of L-sentences A with M A.

(Note that elementary equivalent models are precisely models with the same theory.)

Consider the language of arithmetic. Its intended model is called $\mathscr{A}$. Are all models of the theory of arithmetic isomorphic to the intended model?

No. We can show this by compactness. First, extend the language by a new constant c. Add $c \neq 0$ and $c \neq 1$ etc. Every finite subset is satisfiable in a model based on $\mathscr{A}$, interpreting c as a sufficiently large number. So the whole theory is satisfiable in some model M. This model must have an extra element besides the numbers. Now discard the interpretation of c from the model. (This is called the *reduct* of M to the original language.) The resulting model is still a model of $Th(\mathscr{A})$. But it has an extra element. It is called a *non-standard model of arithmetic*. And yet it is elementarily equivalent to $\mathscr{A}$! There is no sentence in the original language that distinguishes the two models.

Exercise 3.8

Show by compactness that if Γ is a set of first-order sentences with arbitrarily large finite models, then Γ has an infinite model.

The Skolem-Loewenheim theorem makes things much worse.

Are these sentences true?

The second sentence

Consider $\exists x \exists y(x \neq y)$. Any model of this must have at least size 2. Its negation only has models with size 1.

Exercise 3.9

Find a sentence with model size at least 3 and at most 2.

Here is a set of sentences with only infinite models:

xxxx [e.g. BBJ 138]

Corollary: Every satisfiable set of sentences has a model whose domain is a set of natural numbers.

Here are some easy consequences of the completeness theorem.

Definition 3.9

A set of first-order sentences is *satisfiable* if it has a model.

Compactness (Gödel 1929)

If every finite subset of a set is satisfiable then the set itself is satisfiable.

Proof. Let Γ be a set of first-order sentences. Assume for contraposition that Γ is not satisfiable. So $\Gamma \Vdash \bot$. By the Compactness Theorem, it follows that $\Gamma \vdash \bot$: there is a deduction of $\bot$ from Γ. This deduction can only use finitely many sentences from

Γ. So there is a finite subset Γ_0 of Γ for which $\Gamma_0 \vdash \bot$. By the Soundness Theorem, $\Gamma_0 \Vdash \bot$.

Corollary 3.1

If $\Gamma \Vdash A$ then there is a finite subset Γ_0 of Γ such that $\Gamma_0 \Vdash A$.

(Downward) Löwenheim-Skolem

If a set Γ of countable first-order sentences has a model, then it has a countable model.

The Henkin model M_Γ for Γ is countable, as its domain consists of equivalence classes of closed terms in the language, of which (by assumption) there are countably many.

Upward Löwenheim-Skolem

If a set Γ of first-order sentences has an countably infinite model, then it has a model at least as large as any larger cardinality.

Let κ be any uncountable cardinal. Expand the language of Γ by κ new individual constants. For each pair of these constants c_i and c_j, add the sentence $c_i \neq c_j$ to Γ, thereby creating the set Γ^+. If Γ has a countably infinite model M, then all finite subsets of Γ^+ are satisfied in the same model, with the (finitely many) new constants in the set interpreted as distinct objects in the domain of M. By the Compactness Theorem, Γ^+ has a model M^+. All the κ new constants denote distinct objects in this model. So the model has at least κ objects. If we restrict M^+ to the original language of Γ, we get a model of Γ with at least κ objects.

The downward theorem, as stated, was first proved by Thoralf Skolem in 1920, building on an incomplete 1915 sketch by Leopold Löwenheim. The upward theorem was proved by Alfred Tarski in 1935, who also showed that one can find a model whose cardinality is exactly κ.

Let's think about what these theorems mean, starting with compactness. It is easy to find cases where a conclusion is entailed by infinitely many premises, but not by any finite subset of those premises. For example, from the premises:

I like the number 0;

I like the number 1;
I like the number 2;
…

and so on, for all natural numbers, one can infer

I like all natural numbers.

But this conclusion is not entailed by any finite subset of the premises. Similarly, one can infer that some person X is not an ancestor of some person Y from the assumptions that

X is not a parent of Y;
X is not a parent of a parent of Y;
X is not a parent of a parent of a parent of Y;
…

and so on, but not from any finite subset of those assumptions.

The compactness theorem shows that this kind of situation never arises for *first-order logical entailment*. The two inference just describe are valid, but their validity depends on the meaning of non-logical terms: '1', '2', '3', …, and 'number' in the first case; 'parent' and 'ancestor' in the second. From the compactness of first-order logic, it immediately follows that one can't define these concepts using only the connectives, the first-order quantifiers, and identity.

What if we declare the relevant terms to be logical? Let's see what this would mean for a formalized version of the first inference. Take a first-order language with infinitely many constants '0', '1', '2', …, and predicate symbols 'N' and 'L'. The premises can then be formulated as: $L0$, $L1$, $L2$, …. The conclusion is: $\forall x(Nx \to Lx)$. Obviously, the conclusion is not entailed by the premises in the ordinary sense of first-order logical entailment: we can easily find an interpretation of '0', '1', '2', …, 'N', and 'L' that makes the premises true and the conclusion false. But now the idea is that we can define a new kind of entailment – call it "arithmetical entailment" – by fixing the interpretation of the '0', '1', '2', …, and 'N': '0' must denote the number zero, '1' the number one, …, and the extension of 'N' must be the set of natural numbers. Some premises Γ "arithmetically entail" a conclusion A if any model of Γ that conforms to these rules is a model of A. Now $\{L1, L2, ...\}$ arithmetically entails $\forall x(Nx \to Lx)$. But no finite subset of $\{L0, L1, ...\}$ arithmetically entails $\forall x(Nx \to Lx)$. So arithmetical entailment is not compact. Since compactness is a consequence of soundness and completeness, it follows that there can't be a sound and complete proof system for arithmetical validity. That is, there is no way to spell out axioms and rules of inference that define a proof relation $\vdash_{\mathscr{A}}$ so that $\Gamma \vdash_{\mathscr{A}} A$ iff Γ arithmetically entails A – as long as proofs remain finite.

Exercise 3.10

Explain how the argument can be run with a single non-arithmetical constant c instead of the predicate symbol L.

Exercise 3.11

Consider the "proof system" that has all arithmetical truths as axioms, in addition to the axioms and rules of first-order predicate calculus. Can all arithmetical truths be proved in this system? Why doesn't it follow that the system is complete?

Thus compactness reveals a deep expressive limitation of first-order logic. It shows, for example, that one cannot express the concept of finiteness. That is, there is no first-order sentence that is true in all and only the models with a finite domain. [Explain.]

Exercise 3.12

Can you find a sentence that is only true in infinite models? Can you find a sentence that is true in all and only the infinite models?

Here's another example, from the theory of orders.

A relation $\leq$ on a set D is a *linear order* if it is

- *reflexive*: for every x in D, $x \leq x$; - *transitive*: if $x \leq y$ and $y \leq z$ then $x \leq z$; - *anti-symmetric*: if $x \leq y$ and $y \leq x$ then $x = y$; - *total*: for every x, y in D, either $x \leq y$ or $y \leq x$.

A linear order $\leq$ is a *well-order* if every non-empty subset of D has a $\leq$-least element. That is, every non-empty subset S of D has some element $x \in S$ such that for every $y \in S$, $x \leq y$. For example, the less-than-or-equal relation $\leq$ on the natural numbers $\mathbb{N}$ is a well-order: any set of natural numbers has a least element.

Obviously, the conditions of reflexivity, anti-symmetry, and totality can be expressed in first-order logic. So we one can write down a set of first-order sentences with a predicate symbol '$\leq$' that is true in a model iff the model interprets '$\leq$' as a linear order on its domain. We say that these sentences *axiomatize* the concept of a linear order. Can we also axiomatize the concept of a well-order?

The answer is no, because of compactness. Suppose for reductio that some set of first-order sentences Γ is true in a model iff it interprets '$\preceq$' as a well-order. In particular, Γ is true in the model M whose domain are the natural numbers $\mathbb{N}$ and that interprets '$\preceq$' as the less-than-or-equal relation on $\mathbb{N}$. Now let $c_1, c_2, \ldots$ be a list of individual constants. Construct an extension Γ^+ of Γ by adding, for each c_i, the sentence

$$c_{i+1} \preceq c_i \land c_{i+1} \neq c_i,$$

which we may abbreviate as

$$c_{i+1} \prec c_i.$$

Every finite subset of Γ^+ is still satisfiable: it is true in the model M just mentioned. By compactness, Γ^+ has a model. But that model must have an infinite descending chain $\ldots c_{i+1} \prec c_i \ldots \prec c_1$. So it is not a well-order.

Compactness is a fundamental theorem in model theory. Like Gödel, we've derived it from completeness, but note that it has nothing to do with proofs. It's a purely model-theoretic result, that can also be proved without any reference to proofs.

In fact, we can use a version of our Henkin proof of Completeness. Call a set Γ *finitely satisfiable* if every finite subset of Γ is satisfiable. Assume that Γ is finitely satisfiable. We would show that Γ can be extended to a maximal finitely satisfiable set Γ^+ in a language $\mathfrak{L}^+$ with new constant symbols. The Henkin model of Γ^+ is then a model of Γ.

Exercise 3.13

A proof system $\vdash$ is called *weakly complete* if it can prove all validities: if $\Vdash A$ then $\vdash A$. In the previous section, we've shown that first-order logic is *strongly complete*: if $\Gamma \Vdash A$ then $\Gamma \vdash A$. Strong completeness obviously implies weak completeness. We've seen that it also implies compactness. Show that, conversely, weak completeness and compactness together imply strong completeness, given that $\vdash$ has the properties ID and MP.

4 Theories

4.1 Formalized theories

So far, we've talked about pure logic. We often want to consider formalized theories.

For most of history, people did math in an informal manner, relying on a collection of techniques for solving particular problems, and appealing to intuition where something seemed obvious (e.g. that if x > y and y > z then x > z).

In the 19th century, math went through a phase of replacing this with a more rigorous approach. Especially in real analysis. Weierstrass, Dedekind, and others provided precise definitions of limits, of differentiation, of the real numbers themselves, and so on. They formalized the precise assumptions that were needed in an axiomatized system, so that one can prove that such-and-such methods are sound. This was seen as a great step forward.

At the same time, more powerful mathematical theories were developed, such as Cantor's theory of sets, formalized by Zermelo and Fraenkel, or Frege's logic, Russell and Whitehead's Principia, etc. It seemed that all of maths could be unified in such a theory. This would again be great progress.

Formally, a *theory* is a set of sentences that is closed under entailment, so that it contains everything that is entailed by it. We write $\vdash_T A$ or $T \vdash A$ for $A \in T$. A theory is usually given by specifying some sentences, which are called the axioms of the theory. It is understood that such a theory contains every sentence entailed by the axioms.

4.2 Formal arithmetic

Let's think about how to axiomatize the natural numbers.

We don't want infinitely many primitive terms. We can use 0 and s as primitive.

We need to say something about the structure of the natural numbers. Here's an idea:

(4.1) **Q1.** $\forall x \forall y\, (x' = y' \rightarrow x = y)$

(4.2) **Q2.** $\forall x\, 0 \neq x'$

(4.3) **Q3.** $\forall x\, (x \neq 0 \rightarrow \exists y\, x = y')$

What do models of Q look like? We can't have o → o, because every function expression denotes a total function. So every individual must have a successor. We can't have loops, or branching in any direction. [Exercise!]

But is anything missing? Yes. Can you find non-standard models?

We'd like to say that every number is eventually generated from 0 by application of the successor operation. In second-order logic, we can use the induction axiom:

3. $\forall P\ (P(0) \wedge \forall x\ (P(x) \rightarrow P(s(x))) \rightarrow \forall x\, P(x))$.

In first-order logic, we can use the induction schema:

3'. $\forall x\ (\phi(0) \wedge \forall y\ (\phi(y) \rightarrow \phi(s(y))) \rightarrow \phi(x))$,

where ϕ is any formula with one free variable. For note that all numbers are generated from 0 by the successor operation. So if a property holds for 0 and it holds for any other number *provided that* it holds for its ancestor, then – by induction – it holds for all numbers. That's just what Ind says.

This was, in essence, Dedekind's 1888 proposal. Hamlin sec.1.8

You may note that Ind is a schema. We can't actually list all instances of it. That's OK. At least we can mechanically recognize whether something is an instance of an axiom or not. The point about "mechanical" is this: Suppose I declared that the axioms of my theory are all and only the true sentences of arithmetic. That wouldn't count as a genuine axiomatization. I may have introduced a theory alright, but I haven't presented an axiomatized theory.

The language we have at the moment has just 0 and s. Can we define other expressions, like addition?

You can't define them explicitly, in the way we defined   in terms of .

Here's an idea.

Define x+0 = x, and x+s(y) = s(x+y).

The idea is to define an operation by describing how it applies to 0 and then how it proceeds by describing how its value for s(y) is related to its value for y. This is known as a *recursive definition*.

Dedekind proved that it fixes the operation on the natural numbers uniquely: if two operations satisfy the definition, they must yield the same output for all natural numbers as inputs. Otherwise there would have to be a least point of disagreement.

Exercise 4.1

Define multiplication recursively

A problem with recursive definitions is that they are not explicit definitions. The two principles implicitly define addition uniquely, but they don't give us an object-language formula φ(x,y,z) that says that x+y=z.

In second-order logic, we can offer such a formula: x+y=z can be defined as $\forall f[(f(0) = x \land \forall v(f(s(v)) = s(f(v)))) \to f(y) = z]$.

It turns out that there's no way to do this in first-order logic. So we add the two principles as axioms.

https://math.stackexchange.com/questions/449146/why-are-addition-and-multiplication-included-in-the-signature-of-first-order-pea

We do the same with the multiplication axioms.

It turns out that this is enough. E.g., exponentiation *can* be defined explicitly.

Exercise 4.2

Prove $\forall x(s(x) * s(0) = x)$ from the axioms.

Exercise 4.3

Prove the baby axiom $\neg x = 0 \to \exists y(x = s(y))$.

Exercise 4.4

Prove Herbrand's axiom schema: ¬s...sx = x.

Exercise 4.5

Prove x + y = y + x

Intro metalinguistic abbreviations for sssss0. So we can say that PA ⊢ 1+2=3.

Easy to show that PA can prove all particular truths about + and times.

The intended interpretation of PA. This seems to be a particular model. PA clearly true in it.

We've described models with extra numbers for the theory with just axioms 1-3. Does first-order PA still have such models?

Yes, by compactness!

There's no way to avoid this. Even if we added all truths of arithmetic, we'd have such non-standard models.

In practice, the unintended models may not be a problem. We may hope, e.g., that PA contains all arithmetical truths, and isn't that all that matters?

Let Ω be the set of all sentences true in the model. This is a theory, although not given by axioms. We may hope that $\Omega = PA$.

4.3 Set theory

Informally, a *set* is simply a collection of things, which are called the *elements* or *members* of the set. For example, the number 7 is an element of the set $\mathbb{N}$ of natural numbers: $7 \in \mathbb{N}$.

Early set theorists reasoned informally about sets. They took for granted that for any predicate P, there is a set $\{x : Px\}$ of all things that satisfy that predicate. But this leads to paradox, as Cantor first noticed in 1895. The simplest instance is Russell's paradox, discovered at around 1900 by Russell and Zermelo (independently).

Consider the predicate $x \notin x$. If we assume that there is a set of all things that satisfy this predicate, we can call it R. Then, we can ask whether R is a member of itself. If it is, then by the definition of R, it must not be a member of itself. If it isn't, then by the definition of R, it must be a member of itself.

There's something odd about the idea that a set could contain itself. One imagines sets as abstract "containers", and a container can't contain itself. From this perspective, there is no set of all sets, because that would be a set that contains itself. So we really can't rely on the comprehension schema.

Zermelo suggested an attractive alternative. We should think of the sets as built in stages or layers. We start with things that are not sets, called *individuals* or *urelemente*. At the next level, we have all sets of those things. Then we have all sets whose elements occur in stages 0 and 1, and so on. ("And so on" means more than you might have thought here.)

The resulting structure is known as the cumulative hierarchy.

For purely mathematical applications of set theory, it turns out that we can dispense with individuals. There's then nothing at stage 0. At stage 1, there is the empty set. The construction goes like this:

- stage 0: $V_0 = \emptyset$

- stage k+1: $V_{k+1} = \{x : x \subseteq V_k\}$

This goes on forever. But we don't stop there. "After" all the finite stages (of which there are infinitely many), there is a stage ω:

- stage ω: $V_\omega = \bigcup_{k<\omega} V_k$

From there, we can continue to build the sets at stage ω+1, ω+2, and so on. And then we can take the union of all those sets to get the set at a stage we might label ω+ω, or 2ω. After infinitely many more stages, we reach stage 3ω, then 4ω, etc. There are infinitely many of such stages, reach representing an infinite step past the previous one. And we're not done there. After all these stages, there is another stage $\omega \cdot \omega$, or ω^2, where we take the union of all previous stages. And we keep going. Repeating the whole procedure, we reach ω^3, then ω^4, and so on. You might have guessed how this continues. After all these stages comes a stage ω^ω. And we keep going. Much later, we come across stages ω^{ω^ω} and stages with arbitrarily high towers of ω. And we're still only near the bottom of the hierarchy. The set-theoretic hierarchy is *vast*.

There are many ways of formalizing the cumulative hierarchy. The most popular is Zermelo-Fraenkel set theory with Choice (ZFC).

The signature is $\{\in\}$.

Here are some useful abbreviations:

- $x \subseteq y$ means $\forall z(z \in x \to z \in y)$
- $x \cup y$ means $\{z : z \in x \vee z \in y\}$
- $x \cap y$ means $\{z : z \in x \wedge z \in y\}$
- $\bigcup x$ means $\{z : \exists y(y \in x \wedge z \in y)\}$, the union of all sets in x.
- $P(x)$ means the power set of x, i.e. $\{y : y \subseteq x\}$.

Here are the axioms:

1. **Extensionality** $\forall x \forall y[(\forall z(z \in x \leftrightarrow z \in y)) \to x = y]$ This says that a set is completely determined by its elements. There aren't two sets with the same elements.
2. **Empty Set** $\exists x \forall y(y \notin x)$ There is an empty set (the starting point of the hierarchy).

3. **Pairing** $\forall a \forall b \exists x \forall y[y \in x \leftrightarrow (y = a \vee y = b)]$ For any things a,b in the hierarchy, there is a set (higher-up in the hierarchy) $\{a, b\}$ that contains exactly those things. (Note that a, b can be the same thing. In this case, the axiom tells us that there is a set $\{a, a\} = \{a\}$ that contains exactly a.)

4. **Union** $\forall x \exists u \forall y(y \in u \leftrightarrow \exists z(z \in x \wedge y \in z))$ If we have a set x of sets, we may form a set from all the elements of those sets, i.e. the union $\bigcup x$.

   [Query: Can you figure out why this is called the axiom of union? Write up an example, where a is a set of three sets and each of those three sets has two elements. What does the set whose existence is guaranteed by this axiom look like?]

5. **Power Set** $\forall x \exists p \forall y(y \in p \leftrightarrow y \subseteq x)$ If we have a set x, we may form the set of all its subsets, i.e. the power set P(x).

6. **Infinity** $\exists x[\emptyset \in x \wedge \forall y(y \in x \rightarrow y \cup \{y\} \in x)]$ This is needed to show that there are infinite sets.

7. **Separation (Subset Comprehension) – axiom *scheme*** For every formula φ(u,…,p) with parameters p $\forall a \exists b \forall y(y \in b \leftrightarrow [y \in a \wedge \phi(y, \ldots, p)])$ Each stage may carve out *definable* subcollections of earlier sets; no genuinely new elements appear.

8. **Replacement – axiom *scheme*** For every formula φ(x,y,p) that is functional in y $\forall a[(\forall x \in a \exists! y \phi(x, y, p)) \rightarrow \exists b \forall x \in a \exists y \in b \phi(x, y, p)]$ If each member of a set a is sent by a definable rule to some object, that whole image already sits at a later stage. More informally, if f is a function then the image of a set under f is also a set. This is what allows limit stages: take unions of earlier outputs without leaving V.

9. **Foundation (Regularity)** $\forall x[x \neq \emptyset \rightarrow \exists y \in x(x \cap y = \emptyset)]$ No infinite ε-descending chains; every membership walk eventually hits ground. Mirrors the stage construction: nothing can occur before its elements.

10. **Choice (AC)** $\forall F[(\forall u \in F)(u \neq \emptyset) \rightarrow \exists c \subseteq \bigcup F(\forall u \in F)(|u \cap c| = 1)]$ Once an entire family of non-empty older sets is present, a later stage may pick a representative from each.

Axioms 2, 3, 4, 5, 6, 7, 8 are instances of Comprehension: they say that certain ways of identifying sets really succeed in identifying a set.

[Need to explain that functions are sets of ordered pairs?]

One can replace the schemas by second-order axioms to get second-order ZFC.

Technically AC is independent of the cumulative picture, and it has created a great controversy. But it is often convenient in mathematical practice.

Exercises:

Exercise 4.6

Infinity says that there is a set x of a certain kind. List 3 members of this set.

Exercise 4.7

Show that if for any three things a,b,c, there is a set {a,b,c}.

4.4 Sets and numbers

Remember the axiom of Infinity. It draws our attention to an infinite sequence: $\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots$.

This sequence has the structure of the natural numbers. We can think of the empty set as 0, and the next set as 1, and so on.

It's not hard to define addition and multiplication on the finite ordinals, so that e.g. $\{\emptyset\} + \{\emptyset, \{\emptyset\}\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}$. One can then derive the first-order Peano axioms (PA) in ZF.

The details are a little fiddly. We'll skip them.

This means that we can *interpret* PA in ZF. In other words, we can define a structure in ZF that demonstrably satisfies the axioms of PA. In that sense also, ZFC is *stronger* than PA.

(Indeed, it turns out that ZFC can prove purely arithmetical statements that PA cannot prove; e.g. Con(PA).)

The cumulative hierarchy contains a set of all our numbers. This set is called ω. (Yes, like the stage at which it is formed. This is also true for the other numbers: $\emptyset$ is formed at stage $\emptyset$, $\{\emptyset\}$ at stage 1, and so on. [check for fence-post error])

ω isn't the successor of any number. But it has something in common with the numbers: it is "transitive and $\in$-well-ordered". A set is *transitive* if every member of the set is a subset of the set. A set is *$\in$-well-ordered* if every non-empty subset has an $\in$-least element, i.e. an element x for which there is no element y such that $y \in x$.

Definition 4.1: Ordinal

An *ordinal* is defined as a transitive well-ordered set.

The ordinals start with 1,2,3,... but keep going. A *limit ordinal* is an ordinal that is not the successor of any ordinal. The first limit ordinal is ω, the next is 2ω.

We can use ordinals to measure the size of sets. Intuitively, the size – formally, the *cardinality* – of a set is the number of its elements. But what does this mean for infinite sets?

We say that two sets have the same cardinality if there is a bijection between them, i.e. a one-to-one correspondence between their elements.

Definition 4.2: Enumerable Sets

A set is called *enumerable* if it has the same cardinality as the natural numbers $\mathbb{N}$.

Example: the set of pairs of numbers is enumerable. Example: the set of formulas of L_A is enumerable.

But not all sets are enumerable.

Cantor's Theorem

The set of all sets of natural numbers is not enumerable.

> Assume that the set of all sets of natural numbers is enumerable. Then we can list them as $S_1, S_2, S_3, \ldots$. Now consider the set $T = \{n \in \mathbb{N} : n \notin S_n\}$. This set T cannot be in the list, because if it were, then there would be some S_k such that $T = S_k$. But then we have two cases: either $k \in T$ or $k \notin T$. If $k \in T$, then by the definition of T, we have $k \notin S_k$, which contradicts the assumption that $T = S_k$. If $k \notin T$, then by the definition of T, we have $k \in S_k$, which again contradicts the assumption that $T = S_k$. Thus, no such enumeration can exist, and the set of all sets of natural numbers is not enumerable.

This line of argument is an instance of *diagonalization*, which we'll encounter again and again. Think of the sets of numbers as rows in a table. Cantor's set T is constructed from the diagonal of this infinite table. It is the *antidiagonal* of the enumeration, as it is defined to differ at each point from the diagonal. [This needs tidying.]

Cantor's theorem states that the cardinality of the powerset of a set is strictly greater than the cardinality of the set itself. In other words, for any set A, there is no bijection between A and its powerset $\mathscr{P}(A)$.

We can now identify the cardinality of a set with the least ordinal that is equipollent to the set.

The finite cardinals are just the finite ordinals.

There are many different infinite cardinals.

The infinite cardinals are known as alephs. ω is $\aleph_0$, the first infinite cardinal. The next one is $\aleph_1$. The next one is $\aleph_2$, and so on. There's also $\aleph_\omega$. There's an aleph for each omega.

Cantor conjectured, but was unable to prove, that there is no cardinality between $\aleph_0$ and $|P(\aleph_0)|$, i.e. that every set of real numbers is either countable or has the same cardinality as the continuum.

In 1938, Gödel showed that the continuum hypothesis is consistent with ZFC (assuming ZFC itself is consistent). In 1963, Paul Cohen showed that the same is true for the negation of the continuum hypothesis.

Oddly, this straightforward hypothesis about sets of real numbers cannot be answered!

Can we settle the continuum hypothesis in second-order ZFC? It isn't derivable. But if we use a particular set theory to study models of second-order ZFC, then all of them will decide the CH within ZFC2, but how they decide them depends on the ambient set theory!

For set theory, categoricity is a delicate matter anyway. Categoricity means that all models are isomorphic, but models are set-theoretic objects, with sets as their domains. In a sense, every model of set theory (first or second order) is a non-standard model!

4.5 Limitations

Imagine travelling back to the 1920s. The axiomatic approach has been a great success. We had precise theories, that could be interpreted in set theory. A common view was that some formalized set theory like ZF or ZFC could serve as the unifying framework for all areas of mathematics. All of maths would be reduced to a single conceptual primitive, $\in$, and a simple set of axioms.

One wrinkle in this beautiful picture was already known (emphasized especially by Thorald Skolem). Set theories like ZF or ZFC are easy to understand syntactically, as collections of symbols. But how should we understand their meaning?

The formal study of meaning, model theory, is itself standardly couched in set theory. A *model* of ZFC would consist of a set of things over which the quantifiers range, and a relation on that set that is picked out by ∈. But the quantifiers are supposed to range over all sets, and there is no set of all sets! In that sense, every model of ZFC is a non-standard model.

Worse, the language of ZF is countable. It follows by Skolem-Loewenheim that ZF has countable models! But how can a theory which says that there are uncountably many things be true in a model with only countably many things? This is ”Skolem's Paradox”.

It's not a real paradox. 'uncountable' is a defined term. A set is countable iff there is an injective function from the set to the finite ordinals. According to ZF there are sets for which there is no such function. In the countable non-standard models, there are such functions – objectively speaking – but they are not in the domain.

But what all this illustrates is that ZFC doesn't seem to pick out a unique structure. Far from it.

Things don't look much clearer if we go second-order, as long as the second-order quantifiers are interpreted as ranging over sets. A model still consists of a set of things over which the quantifiers range. The second-order quantifiers range over sets of these things. These sets are sets according to the ambient theory that describes the models. They aren't among the things the interpreted theory would call ”sets”. Which second-order statements are true now depends on the modelling set theory.

For set theory, categoricity is a delicate matter anyway. Categoricity means that all models are isomorphic, but models are set-theoretic objects, with sets as their domains. In a sense, every model of set theory (first or second order) is a non-standard model!

This also suggests that the higher-order characterisations or numbers etc. that seem categorical may not really be pinning down the relevant structures, if we adopt a set-theoretic interpretation of higher-order logic. We'd need to ensure that the ambient set theory has a definite interpretation.

A way to resolve this would be to give a non-set theoretic interpretation of higher-order logic.

Move rest to ch/5?

Many mathematicians were not too troubled by these issues. They seem too philosophical, irrelevant to mathematical practice, which is about what can be derived from what.

We'd like to ensure the following:

- Completeness: every statement in the language can be proved or disproved from the axioms.

Hilbert wanted a unified complete arithmetic and geometry etc, so that no appeal to intuition is required any more. Maths studies certain structures that are defined by axioms.

Distinguish this from ”completeness” of a logic.

Note that categoricity does not guarantee completeness. It does so only if the background logic is complete.

Conversely, non-categoricity does not entail incompleteness. The unintended models may not be an issue.

Another thing we want to ensure:

- Consistency: no contradiction can be proved from the axioms.

Consistency of PA is not really an issue, but for strongly infinitary theories like ZFC it is.

This was important because there were doubts about the consistency of the new theories. For example, Russell’s paradox showed that naive set theory was inconsistent.

Hilbert realized while the new theories themselves dealt with highly infinitary matters, their consistency was a finitary topic: we need to show that no contradiction can be proved from the axioms, and proofs are finite objects. So there’s hope of being able to prove that ZFC is consistent.

Remember also that we could prove that FOL and PROPCAL are consistent, by proving that they are sound. One might try a similar proof for PA. I.e., we’d specify a model and show that (1) all axioms are true in that model, (2) the rules preserve truth in the model.

Godel shattered this program.

5 Computability

5.1 The Entscheidungsproblem

Move to beginning of ch.5?

Suppose we wonder whether a sentence is valid.

We can tackle this question from two directions. We could try to construct a proof. From the other direction, we could try to construct a countermodel.

Completeness means that if the target sentence is valid then there *is* a proof. And obviously, if the target sentence is invalid then there is a countermodel. But will we able to find one or the other? Is there an algorithm for doing so – in principle?

This is the Entscheidungsproblem.

In general: A property P of strings is said to be decidable if ... there is a total Turing machine that accepts input strings that have property P and rejects those that do not.

Here's an idea. Go through all possible proofs, with increasing length. If there is a proof of the target sentence, you'll eventually hit it.

This means that there is an algorithm for establishing that a sentence is valid if it is valid. (This algorithm is horribly inefficient. We can do better. But the task is NP-complete.)

This algorithm doesn't return anything if a sentence is invalid. It just runs forever.

Exercise 5.1

What if we simultaneously try out all models, with increasing size?

Models can be infinite. If a sentence has only infinite countermodels, our algorithm will never find them.

It turns out that there is NO algorithm for deciding whether a sentence is invalid. This is called the *undecidability* of predicate logic.

To prove this, we need a general concept of algorithm. It won't do to just show that a given algorithm doesn't do the job. The problem was raise in 1928 by Hilbert and Ackerman, and prompted a search for a general concept of algorithms or computations.

This concept was found in 1936. With it, the Entscheidungsproblem could be answered, negatively.

5.2 Preview Gödel

Imagine it is 1930. First-order logic has been rigorously developed. We have axiomatized theories. On a structuralist or formalist view, the axioms define the subject matter. From now on, no mysterious intuitions or questionable steps are involved any more when proving mathematical claims. Moreover, we can formally prove consistency of infinitary theories. Or so it seems.

What is needed to show that this programme works?

1. Prove completeness of axiomatized theories.
2. Prove consistency of axiomatized theories.
3. Solve Entscheidungsproblem.

Suppose something is entailed by the axioms, but there is no way to show that it is, without relying on intuition. Then that's not very useful. Gödel's completeness theorem shows that there will always be a proof. But how do you find a proof? Ideally, that, too, doesn't require intuition. Ideally, there's a mechanical way to find proofs, just as there is a mechanical way to add or differentiate.

We'd like PA = Ω. Generally, we'd like completeness.

We'd also like *decidability*. A set of sentences is decidable if there is a mechanical procedure for checking, for any sentence, whether it belongs to the set or not. This would be nice to have for mathematical theories.

Godel shattered this program.

He showed that no mathematical theory of a certain strength, that includes PA and ZFC, can be complete (unless it is inconsistent). More precisely:

Every consistent and (recursively) axiomatized theory in which all computable functions are representable is incomplete.

He also showed that there are serious problems for proving the consistency of any such theory. In particular, one can't prove the consistency of ZFC (or PA) in any theory that's strictly weaker.

Since 2nd-order PA is complete, it immediately follows that 2nd-order logic is unformalizable. (This was part of what caused a move towards concentrating on first-order logic and first-order theories.)

Gödel's original argument also showed that finding proofs is not a process of the same abstract kind as adding numbers, insofar as any algorithm for finding proofs would fall into a different category than familiar mathematical algorithms. Gödel couldn't yet show that there is *no* such algorithm at all. This requires having a fully general theory of algorithms or computations.

In the next 3 chapters, we'll explore this theory, as it emerged in 1934-1936. That task is important not just for the Entscheidungsproblem, but also for Gödel's incompleteness theorems.

The crucial feature of proofs is that they are finite and verifiable, in the sense that one can mechanically check whether something is a proof or not.

The trick will be to code proofs as numbers, and find an arithmetical predicate that captures that a code number represents a proof of a given formula.

From: https://math.stackexchange.com/questions/2205943/godels-incompleteness-theorem

Then, show the basic idea of arithmetization, i.e. a way of assigning numbers to logical symbols, expressions, and derivations, so that statements about arithmetic can be treated as statements about logical statements and proofs. Just show some very basic examples of how such a coding scheme could work, and then just wave your hands and say that we can create a formula that effectively says 'the statement with Godel number x is not provable from A', where the fact that we can have such a formula depends on A being 'at least as strong as PA' and being finite.

Next (and again without any further proof) mention the Diagonal Lemma through which Godel established that for every formula there is a sentence that is true if and only if that formula is true for the godel number of that sentence. Thus, in particular, there will be a Sentence G (the Godel sentence) that ends up saying "I am not provable from A"

And now do the grand finale! If G is false, then G is provable from A, but given that A is sound, that means G is true. Ok, so G cannot be false, so it is true. And so G is indeed not provable from A. So, there is something that is true, but not provable from A, and thus A is not complete.

This trick is very general. It works for all computable relations and functions. Before we get to it, let's talk about what computable relations and functions look like. This is independently important for maths and computer science in logic. Remember for example the Entscheidungsproblem.

5.3 Computable functions

Copy from Machover

In high school, you learned how to calculate answers to mathematical questions: how to do sums, divisions, derivatives, etc. You learned an algorithm that could in principle be taught to a machine.

We might wonder if all mathematical questions can be answered with some such algorithm. Leibniz envisaged this.

Let's be clear about what that means.

In a sense, it's trivial that for every mathematical question there's an algorithm that gives the answer: simply write down the answer. No calculation required.

This approach doesn't scale. We want to compress what the machine does. An algorithm is an instruction to find the answer to every question of a certain type, which typically has infinitely many instances. That is, we're interested in the computability of *functions*.

And by computability, we mean computability with an effective procedure, following a pre-defined set of exact rules – rules that can in principle be carried out by a computer.

Definition 5.1: Effective Computability

A function is *effectively computable* if there are precise, determinate instructions for computing the output for any given input.

We don't assume that anyone is actually able to follow the instructions in practice. Doing so might require more time or memory than anyone has. Multiplication, for example, is effectively computable, but nobody can in fact compute the product of any two numbers.

I haven't said exactly what it means that a function is effectively computable. Different precise definitions have been considered. We'll consider two in the next two chapters: computable by a Turing machine and recursivity. They will turn out to be equivalent in the sense that they define the same class of functions.

All plausible proposals for making precise the notion of effective computability turn out to be equivalent.

[intro here so that we can make labour-saving uses of it.]

Church's Thesis posits that the intuitive notion of ”computable” corresponds exactly to the functions that are μ-recursive.

We've shown that these are precisely the functions that are computable by a Turing machine.

To show that there is no effective procedure for solving some problem – say, the Entscheidungsproblem –, one generally needs to appeal to Church's Thesis in order to connect the informal notion of "effectively computable" with a formal notion of computability.

Besides these *unavoidable* appeals to Church's Thesis, we'll also frequently make *avoidable* or *lazy* appeals to Church's Thesis when we need the assumption that some particular function is recursive. If the function is obviously computable, we'll sometimes say that it is recursive "by Church's Thesis". In each such case, one could directly prove that the function is indeed recursive, without using Church's Thesis.

5.4 The halting problem

Any function you can think of is computable. Are there uncomputable ones?

Yes: there are more functions than computations.

But can we give an example?

An algorithm is a recipe. Presumably every algorithm can be stated in a suitably general language, in finitely many symbols. So one could define an algorithm to list all the algorithms. But now we can write another algorithm that goes as follows: for any input n, list the algorithms up to n, then compute the algorithm with input n, and return the output plus 1.

This algorithm can't be on the list! What's wrong with it?

Perhaps we can't enumerate the algorithms? No. The problem is that algorithms may run forever. That's not enough to escape the paradox. We must also assume that we can't detect whether an algorithm halts.

This gives us an uncomputable function!

Exercise 5.2

Let the *power* of a formula be the size of its smallest model. Consider the function that maps any number to the largest power of a formula of that length. Explain why this function is not computable, given that first-order logic is undecidable.

[Maybe move to after Turing section, where we could prove undecidability?]

We already know from the unsolvability of the halting problem that there is no complete maths: Whether a Turing Machine halts on a given input is a mathematical question.

(It's not a question in engineering!) If there was a complete axiomatic theory of Turing Machines, we could try to derive that M halts on input x, while also trying to derive that M doesn't halt. One of these would yield a positive answer in a finite amount of time.

Exercise 5.3

I often see the halting problem misconstrued as "it's impossible to tell if a program will halt before running it." This is wrong. The halting problem says that we cannot create an algorithm that, when applied to an arbitrary program, tells us whether the program will halt or not. It is absolutely possible to tell if many programs will halt or not.

5.5 Decidable sets and relations

Definition 5.2: Decidable Sets

Terminology: a set is *(effectively) decidable* if there is an algorithm that can determine whether something is in the set or not.

Exercise 5.4

Is the set of even numbers decidable? Is the set of prime numbers decidable?

A set is decidable iff its characteristic function is effectively computable.

Note that finite sets are always decidable.

If we allowed the rules/programs to be infinite, even infinite sets would be decidable. We don't.

We can generalize the concept of decidability to relations. A binary relation is decidable if there is an algorithm that can determine whether it applies to any given pair.

5.6 Recursive enumerability

Definition 5.3: Effectively Enumerable Sets

A set is *effectively enumerable* if there is an algorithm that can enumerate its elements, i.e., list them one by one. Equivalently, there is an effectively computable function that outputs 'yes' for every element in the set, and never for an element not in the set.

Similarly for relations.

Effectively enumerable sets are also called *semidecidable* or *recursively enumerable* – for short, *r.e.*.

Some sets are undecidable but r.e. For example, the set of all Turing machines that halt on a given input is r.e. but not decidable. We can enumerate the machines that halt, but we cannot decide whether a given machine halts.

R.E. from Decidable Relations

If R is a decidable (two-place) relation, then the set of all x such that $\exists y R(x, y)$ is r.e.

To check whether a number n is in the set, we compute $R(n, 0), R(n, 1), R(n, 2)$, and so on. If n is in the set, we will eventually find this answer. (If it isn't, we won't.)

The converse is true as well:

Decidable Relations from R.E.

If S is r.e. then there is a recursive relation R such that $x \in S$ iff $\exists y R(x, y)$.

If S is r.e. then there's a machine that enumerates S. The relation R is the relation that holds between x and y iff the machine has produced y among the first x items.

This second proposition can be used to formally define the concept of r.e.

Kleene's Theorem

If a set and its complement are both r.e. then the set is recursive.

BBJ 82.

6 Turing computability

6.1 Turing machines

Follow IGT ch.41?

A Turing machine is an idealized machine.

A Turing machine works on a tape that is unbounded in both directions. The tape is divided into cells, each of which is either blank or hold a symbol from some finite alphabet.

The machine works in discrete steps. Each step operates on a single cell at a time. At the start of a step, the machine *reads* the symbol in the cell. It then erases that symbol and either leaves the cell blank or writes a new symbol onto the cell. (The new symbol might be the same as the old symbol.) Finally, the machine moves its head one cell to the left or right.

To program a Turing machine, we need to give it instructions for which action to take depending on the content of the cell it is currently reading. For example, an instruction might say ”if the cell contains a 1, erase it and move right; if the cell is blank, write a 1 and move left”.

A single instruction of this kind would not allow the machine to perform any interesting computation. Instead, we need a set of instructions that can be applied in sequence. For example, we might program a machine to first follow the above instruction and then a different instruction: ”if the cell contains a 1, erase it and move left; if the cell is blank, write a 0 and move right”.

Of course, the machine isn't really reading English instructions. To implement such a set of instructions, the machine must be able to go into different internal *states* in which it responds to the scanned symbol in different ways.

The machine begins in a *starting state*. We program what it should do, including possibly that it should go into a different state.

Here is a simple example. q1 is the start state.

For most computational tasks, it isn't necessary to keep track of all the past symbols and acts. This means that we can allow the machine to return to an earlier state.

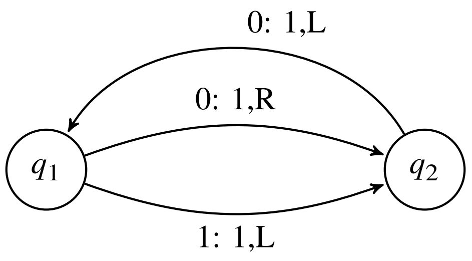
We can simplify the above algorithm.

We could say that the machine EITHER changes the cell content OR moves. Some authors do this. The definitions are equivalent.

We can also restrict the number of symbols. It turns out that it makes no difference whether there is one symbol or two symbols, or any finite number. A common assumption, that we'll follow, is that there is only one symbol, a ”stroke”.

The *program* of a Turing machine can be represented as a flow chart.

Here is one.



under its head, then it writes a new symbol in that cell, and finally it moves the head one cell to the left or right.

We'll assume that there is only one symbol, a ”stroke”, so that each cell on the tape is either blank or contains a stroke.

The tape is infinite, so it can be thought of as a sequence of cells indexed by integers, with the leftmost cell being indexed by 0.

The machine has a head that can read and write symbols on the tape and move left or right.

Exercise 6.1

Describe a TM that erases a consecutive string of 1s at whose left end it starts.

6.2 Coding

We've assumed that a TM takes sequences of 1s as input.

Mention the general fact that we need to express questions in a particular string format to make algorithms applicable. When we say that a function is computable, we mean the function from numbers to numbers, but computations require a representation of the numbers.

Some representations of numbers make computations easier. It is easier to compute 34 * 100 in decimal than in Roman. But in principle, it can be done either way.

Suppose, for example, that we define an algorithm on strings of symbols. We need to code the strings as sequences of strokes and blanks.

There are many ways to do this. We can use sequences of strokes alone. In effect, we code the strings as numbers.

The obvious technique is to first code each symbol as a number. Now we need to code sequences of symbols, or equivalently sequences of code numbers. How could we do that?

(As BBJ say, the most important aspect of coding strings is that the concatenation function that takes two code numbers and returns the code number of their concatenation is recursive.)

Suppose we have a countable alphabet of symbols $a_1, a_2, \ldots.$ We assign to each symbol a number: $a_1 = 1, a_2 = 2$, etc.

In L_A, we have infinitely many variables. (Could we simply insist that variables are written v_1, v_2, etc. in decimal notation?)

Let's think about the simplest case: coding pairs of symbols.

In effect, we need to code pairs of numbers as single numbers. Cantor's zig-zag method. The formula for this is $J(x, y) = \ldots.$

We can also code sequences of numbers by exploiting the fact that every number has a unique prime factorization.

Recall that a prime number is a natural number which is greater than 1 and can be divided only by 1 and the number itself (all the other numbers greater than 1 are "composite"). There are infinitely many prime numbers; the beginning of the sequence is 2, 3, 5, 7, 11, 13, 17, …

The fundamental theorem of arithmetic (or the unique-prime-factorization theorem) states that any natural number greater than 1 can be written as a unique product (up to ordering of the factors) of prime numbers.

Let then p1 be the first prime number, p2 the second prime number, and so forth. Given an arbitrary finite sequence of positive numbers (0 would cause complications) with length k+1, $(n_0, n_1, \ldots, n_k)$, it can be uniquely coded as a product of powers of the prime numbers $p_1, p_2, \ldots, p_{k+1}$ as follows: …

Suppose we want to code the sequence 10,4,6. Since the sequence has length 3, we use the first three prime numbers: 2, 3, and 5. We can then code the sequence as

$$2^{10} \cdot 3^4 \cdot 5^6 = 1296000000.$$

To decode the number, we can factor it into primes and read the exponents. In this case, the code number uniquely factors into primes as $2^{10} \cdot 3^4 \cdot 5^6$. This tells us that the number codes a sequence of three numbers, where the first number is 10, the second is 4, and the third is 6.

In practice, this method is terribly inefficient, in part because the code numbers grow large very quickly. But efficiency is not our concern here.

6.3 Uncomputability

Need to code TMs so that we can enumerate them.

https://scottaaronson.blog/?p=8972

What makes BB(748) independent of ZFC is not its value, but the fact that one of the 748-state machines (call it TM_ZFC_INC) looks for an inconsistency (proof of FALSE) in ZFC and only halts upon finding one.

Thus, any proof that BB(748) = N must either show that TM_ZF_INC halts within N steps or never halts. By Gödel's famous results, neither of those cases is possible if ZFC is assumed to be consistent.

Turing noted that one can prove the Turing-undecidability of predicate logic, by "reducing the halting problem to it". The idea is that for any TM and number n, we can find a set of FOL sentences Γ and a sentence D such that Γ entails D iff the machine halts when started on input n. So if we could construct a TM that decides entailment, we could construct a TM that solves the halting problem.

The sentences in Γ simply provide a first-order description of the TM and its input. Here we use non-logical symbols for natural numbers, a predicate Q so that $Q(x, y)$ says that at step x the machine is in state y, another predicate @ so that $@(x, y)$ says that at step x the machine is positioned at square y, and a predicate M so that $M(x, y)$ says that at step x there is a mark (a stroke) in square y. One can then easily express the starting configuration of a TM. Likewise, one can formulate each row of the machine table. To these, one needs to add some background information about the natural numbers – PA is more than enough; a finite fragment will do. That's Γ.

The sentence D says that there is a step at which the machine halts. It is a disjunction, each disjunct of which corresponds to a state/symbol combination for which there is no entry in the machine table. For example, if there's no entry in the table for what to do in state 0 when reading a blank, a disjunct of D will be $\exists x \exists y (Q(x, 0) \land @(x, y) \land \neg M(x, y))$.

To complete the proof, one needs to show that the machine does indeed halt iff Γ entails D. This is not hard, but it takes a few pages of labour. We'll give another proof of Church's Theorem later.

Exercise 6.2

Show equivalence between decision problem for entailment and DP for validity and DP for satisfiability.

Exercise 6.3

Using Church's thesis, prove Trakhtenbaum's Theorem.

7 Recursive Functions

7.1 Primitive recursive functions

Remember: an effectively computable function is one for which there is a finite list of precise instructions to determine the output for any given input, without drawing on external resources or creativity.

We will concentrate on functions from natural numbers to natural numbers.

Here's a natural ideal of how to approach the definition of computable functions.

The computable arithmetical functions are composed out of simpler functions, and we can use this composition to compute them.

Addition is repeated successor, and that's how children learn to add, with the counting on strategy.

Similarly, multiplication is repeated addition. This isn't the most efficient way to multiply, but it's an algorithm.

So maybe we can define the computable functions as functions that are in this way definable?

We start some base functions.

Definition 7.1: Base Functions

The first is the *successor function* s, whose value for any input number is the next larger number:

$$s(n) = n + 1$$

The second is the *zero function* z that returns 0 for number given as input.

$$z(n) = 0$$

Finally, we have a supply of *identity* or *projection functions* id_i^n that take some numbers as inputs and return one of those numbers. For example, id_1^1 is the function that takes a single number and outputs that same number. id^2_1 takes two numbers and outputs the first; id^2_2 outputs the second. In general,

$$id^n_i(x_1, \dots, x_n) = x_i$$

These functions are trivially computable, without any sub-computations. We might say that they are computable *in one step*.

Given some computable functions f and g, we can define a new function h by composing them, applying one to the output of the other:

h(x) = f(g(x))

Here we assume that f and g both take one number as input. For the general case, assume that f is a function of m arguments, and each of $g_1, \dots, g_m$ is a function of n arguments. Then we define the *composition* of f and $g_1, ..., g_m$ as:

$$h(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

Instead of introducing a new name 'h' for the composed function, we can also write the composition as $Cn[f, g_1, \dots, g_m]$.

For example, $Cn[s, z]$ is the function that takes a number as input, then passes it to the zero function and passes the output to the successor function. This is the constant function that always outputs 1.

$Cn[s, Cn[s, z]]$ is another function that takes a number, passes it to $Cn[s, z]$, which outputs 1, and then passes that to the successor function, which outputs 2. This is the constant function that always outputs 2.

Evidently, the result of composing effectively computable functions is still computable. To compute $f(g_1, \dots, g_m)$, one only needs to go through the computation of $g_1(x_1, \dots, x_n)$, ..., $g_m(x_1, \dots, x_n)$, and then apply f to the results. The number of steps this takes is the sum of the number of steps needed to compute each $g_i(x_1, ..., x_n)$ plus the number of steps needed to compute f for the result.

In fact, there might be simpler ways to compute the composite function, with fewer steps. So this is the upper bound on the number of steps needed.

Another way of defining functions.

Consider addition, which can be defined in terms of counting, as follows (where free variables are assumed to be universally bound):

$$x + 0 = x$$
$$x + s(n) = s(x + n)$$

This says that to add 0 to a number, you leave the number unchanged; to add a larger number, you add 1 to the result of adding 1 less than that number. We've encountered this pair of equations as part of PA.

Similarly for multiplication:

$$x \cdot 0 = 0$$
$$x \cdot s(n) = x \cdot n + x$$

Two different types of example. First, the factorial:

$$0! = 1 \tag{7.1}$$
$$(k + 1)! = k! \cdot (k + 1) \tag{7.2}$$

Second, the *delta* or *anti-signum* function that takes every positive integer to 0, and 0 to 1:

$$\alpha(0) = 1 \tag{7.3}$$
$$\alpha(k + 1) = 0 \tag{7.4}$$

Definition 7.2: Primitive Recursion

In the most general case, a function h is defined by primitive recursion from two other functions, f and g. f specifies the value of h for the case of 0, and g specifies the value of h for larger inputs based on the value of h for smaller inputs. I.e., we assume the following format:

$$h(x, 0) = f(x) \tag{7.5}$$
$$h(x, s(y)) = g(x, y, h(x, y)) \tag{7.6}$$

(Actually can allow for multiple arguments for x.)
We can compress this into the notation Pr[f,g].

For example, addition is Pr[z,$Cn[s, id_3^3]$]. For
$x + 0 = z(x) = 0$, and $x + s(y) = Cn[s, id_3^3](x, y, x + y) = s(x + y)$.
If the functions f and g are computable, then so is Pr[f,g].
The definition of a pr function tells us how to compute the value for a given input n. To compute h(x,y), we first compute $c_0 = f(x, 0)$ using the function f. Then we compute $c_1 = g(x, 0, x_0)$ to get h(x,1), then $c_2 = g(x, 1, c_1)$ to get h(x,2), and so on until we reach h(x,y). The number of steps needed is the sum of the steps needed to make these computations.
E.g. factorial in JavaScript:

```
function factorial(n) {
    let result = 1;
    for (let i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Exponentiation/Power

$$n^0 = 1 \tag{7.7}$$
$$n^{k+1} = (n^k) \cdot n \tag{7.8}$$

Truncated predecessor

$$\text{pred}(0) = 0 \tag{7.9}$$
$$\text{pred}(k + 1) = k \tag{7.10}$$

Truncated difference

(7.11) $\quad n \dot{-} 0 = n$

(7.12) $\quad n \dot{-} (k+1) = \text{pred}(n \dot{-} k)$

Thus $n \dot{-} k$ is the difference between n and k if $n \geq k$ and is 0 if $n \leq k$.

Definition 7.3: Primitive Recursive Relations

We also want a notion of a relation's being primitive recursive. A suitable notion can be defined in terms of primitive recursive functions as follows. The characteristic function of an m-place relation R is the m-place function χ such that if $R(n_1, \dots, n_m)$ holds then $\chi(n_1, \dots, n_m) = 1$, and if $R(n_1, \dots, n_m)$ does not hold then $\chi(n_1, \dots, n_m) = 0$. We define: a relation is primitive recursive iff its characteristic function is primitive recursive.

E.g. the set of odd numbers is p.r. because its characteristic function can be defined thus: $\chi(0) = 0$, $\chi(k + 1) = \alpha(\chi(k))$.

Exercise 7.1

Show that if a relation is p.r. then so is its negation. Solution: if χ is the characteristic function of R, then the complement of R has characteristic function χ', where $\chi'(n_1, \dots, n_m) = \alpha(\chi(n_1, \dots, n_m))$.

Exercise 7.2

Conjunction – simply multiplication.

7.2 The extent of primitive recursive functions and sets

Additional methods of definition are allowable in defining primitive recursive functions and relations, provided they can be reduced to applications of recursion and composition. Three methods in particular will be of great use to us.

New relations can be defined from given ones by truth-functional (Boolean) combination. For example, k > n iff not $n \leq k$; and $k \geq n$ iff $k > n$ or $k = n$. We claim that any truth-functional combination of primitive recursive relations is primitive recursive.

If is defined by cases from p.r. functions, with conditions whose characteristic function is p.r., then it is p.r. itself.

Composition also gives us the means to capture definition by cases. For example, suppose we wanted to define the function of k and n that yields k^2 if $k \leq n$ and n^2 if $n < k$. We can do this by using addition, multiplication, truncated difference, and switcheroo thus, thereby showing that this function is primitive recursive: $k \cdot k \cdot \alpha(k \dot{-} n) + n \cdot n \cdot \alpha(n + 1 \dot{-} k)$

[BBJ p.74]

If a function is defined by bounded quantification over primitive recursive functions, then it is primitive recursive.

Another definition method we will want to use for relations is bounded quantifi- cation. For example,

k divides n iff $\exists p \leq n(p \cdot k = n)$.

n is prime iff $n > 1 \wedge \forall k \leq n(k \text{ divides } n \rightarrow (k = 1 \vee k = n))$.

It is straightforward to show that if a relation R(k, n) is primitive recursive then so is the relation $(\forall k \leq p)R(k, n)$, which has arguments p and n. Let χ be the characteristic function of R, and define χ' by recursion thus: $\chi'(0, n) = \chi(0, n)$, $\chi'(p + 1, n) = \chi'(p, n) \cdot \chi(p + 1, n)$. Thus χ' is primitive recursive, and is the characteristic function of $(\forall k \leq p)R(k, n)$, since $\chi'(p, n)$ is 1 just in case each of $\chi(0, n), \ldots, \chi(p, n)$ is 1, that is, just in case each of $R(0, n), \ldots, R(p, n)$ holds. Bounded existential quantification can be obtained from bounded universal quan- tification by truth-functional operations

A final definition-method we shall use frequently is bounded leastness. This is used to define a new function from a given relation. The notation we use is this: an expression $(\mu k \leq p)R(k)$ denotes the least number $k \leq p$ such that R holds of k, if there is such a number, and denotes 0 otherwise.

Of special importance later will be the fact that certain operations on code numbers are primitive recursive.

Specifically, suppose x is a number that codes a sequence of numbers $x_1 \ldots x_n$. Then the following functions are p.r.:

len(x) entry(x, i)

Also pr is

seq(x)

(a) Divides(a,b) := $\exists k \leq b(k \cdot a = b)$. (Bounded existential → p.r.)

(b) Pow(p,0)=1, Pow(p,n+1)=Pow(p,n)·p. (Primitive recursion → p.r.)

(c) Prime(x). x is prime iff x>1 and $\forall y < x \forall z < x(y * z \neq x)$.

exp(x,i) is the exponent of the prime p_i in the prime factorization of x.

Intuitively: start at n=0 and go up until $p_i^{(n+1)}$ stops dividing x; the last n for which it did divide is the exponent.

We want the least k such that every prime p_j with $j \geq k$ occurs with exponent 0.

Define an auxiliary predicate

TailZero(x , k) := $\forall j \leq x(j \geq k \to \exp(x,j) = 0)$.

TailZero is a bounded universal statement whose matrix is p.r., hence itself p.r.

Now put

len(x) = $\mu k \leq x$ TailZero(x , k).

entry(x , i) = if i < len(x) then exp(x , i) else 0.

7.3 Minimization

But these can't be all the computable functions. We can diagonalize out.

The antidiagonal is computable, but not primitive recursive.

Somehow, this trick must not work for set of all computable functions. But if we define the functions in terms of applying operations to the base functions, then we can always mechanically enumerate the set. So how could the argument fail?

Think about it.

The argument will fail because sometimes one can't effectively compute the output of a function f_i for a given input i, even though f_i is effectively computable. That's because f_i may be partial: it may be undefined for input i. Moreover, for a tweak to the argument not go through, there must be no general way to tell if a function is defined for some input.

This curious argument shows that a general account of computable functions must include *partial* functions.

Or maybe there's no way of listing the total computable functions. I.e., we can't define all total computable functions in terms of the base functions and operations on them.

Almost any function you can think of is primitive recursive. But some functions are computable but not primitive recursive. We already know one: the antidiagonal of the primitive recursive functions. Another, more directly mathematical example is the Goodstein function. It is defined as follows. (Feel free to skip the details.)

https://risingentropy.com/the-mindblowing-goodstein-sequences/

To explain this function, note first that, for any $n > 1$, any number x can be expressed as a sum of powers of n. For example, $266 = 2^8 + 2^3 + 2^1$. We can push this idea further by writing each exponent as a sum of powers of n as well, until all numbers in the representation are less than or equal to n: $266 = 2^{2^{2+1}} + 2^{2+1} + 2^1$. This is called the ”hereditary base-2 representation” of 266.

Next, we define the ”Goodstein sequence” for a number n. The first item in the sequence is n. For the second item, we replace each 2 in the hereditary base-2 representation of n by 3, and subtract 1. For the third item, we then replace each 3 in the hereditary base-3 representation by 4, and subtract 1. And so on.

Example.

These sequences grow very large very quickly. Surprisingly, however, their growth eventually stalls and reverses, until the process reaches 0, where it ends. This is *Goodstein's Theorem*, proved by Reuben Goodstein in 1944.)

The *Goodstein function* now simply maps any number n to the length of the Goodstein sequence that starts with n. This function maps each natural number n to the number of steps it takes for the Goodstein sequence for n to reach 0. This function is computable, but not primitive recursive.

The function that returns the next item from the previous item in a Goodstein sequence is primitive recursive. But to determine how long it takes for a sequence to reach zero, one must simply go through the items in the sequence, in an unbounded loop.

Definition 7.4: Minimization

Formally, the third operation we need is called *minimization*. The minimization of a function f of $n + 1$ arguments is a function h of n arguments that returns, for any $x_1, \dots, x_n$, the smallest number y for which
(a) $f(x_1, \dots, x_n, y) = 0$, and (b) $f(x_1, \dots, x_n, v)$ is defined for all $v < y$.
If there is no such y, the function returns nothing. So even if f is total, its minimization Mn[f] may be partial.

Exercise 7.3

What is Mn[+]? What is Mn[*]?

Mn[+] is defined only for x=0, in which case it returns 0; Mn[*] is z.

If f is computable then so is Mn[f]: we simply need to compute $f(x_1, \dots, x_n, i)$ for each i starting at 0 until we find a case where $f(x_1, \dots, x_n, i) = 0$, in which case we return i.

7.4 Every μ-recursive function is Turing computable

Copy from IGT sec. 42.2? Or: https://www.cs.utep.edu/vladik/cs5315.21/equiv.pdf

The task is straightforward. We need to show that (i) the base functions are computable by a TM, and (ii) that if some functions are turing-computable then so is any function definable from them by composition, primitive recursion, and minimization.

The zero function. We can describe a TM that returns the output 0 for any input. In unary notation, 0 is the empty tape. See exercise ex-erase.

The successor function.

7.5 Every Turing-computable function is μ-recursive

Copy proof sketch from Smith's Intro, sec. §42.3? But very sketchy.

The converse is more involved but entirely number-theoretic.

Step A Gödel numbering of configurations

• Represent a single configuration C = (q, head, tape) by a natural number $\langle C \rangle$ using a pairing function or prime codes. • Encode an entire TM M as an integer e = $\langle M \rangle$.

Step B The "next configuration" function is primitive recursive

Lemma 1 (Key): there exists a primitive-recursive function NEXT(e,$\langle C \rangle$) = $\langle C' \rangle$ that returns the code of the configuration obtained by applying one transition of machine e to configuration C (or C itself if it is already halting). Proof sketch: look up the current state/symbol in e's transition table (which is encoded in a bounded, hence primitive-recursive, search), perform the symbol rewrite, move the head, and adjust the state field—all by bounded arithmetic.

Step C: Primitive-recursive predicates for "halts in ≤ s steps" Define by bounded iteration

$$\mathrm{CONF}^{0}(e,x) = \langle \text{initial configuration of } e \text{ on } x \rangle \tag{7.13}$$

$$\mathrm{CONF}^{s+1}(e,x) = \mathrm{NEXT}(e, \mathrm{CONF}^{s}(e,x)) \tag{7.14}$$

Both are primitive recursive because NEXT is, and s is a bound.

Then HALT≤(e,x,s) ≡ "state of CONF^{s}(e,x) is halting" is primitive recursive as well.

Step D: μ-operator captures the exact stopping time Use unbounded search: t = μ s. HALT≤(e,x,s) which is defined iff M halts on x. Because the predicate under μ is primitive recursive, the resulting partial function t(e,x) is μ-recursive by definition.

Step E Extracting the output A primitive recursive function RESULT($\langle C \rangle$) returns the contents of the designated output track of configuration C. Compose: Out(e,x) = RESULT(CONF$^{t(e,x)}$(e,x)) where t is from Step D. Therefore Out is μ-recursive and agrees exactly with the function computed by machine e.

Conclusion: for every Turing machine M there is a μ-recursive function f_M that coincides with M's input/output behaviour.

8 Representability

Goldfarb ch.4

8.1 Robinson Arithmetic

We'll now prove Goedel's incompleteness theorem for arithmetic. Why arithmetic? Two reasons. (a) It's a simple, familiar branch of maths. If arithmetic can't be completely axiomatized then obviously more powerful theories can't either. (b) In a sense, arithmetic is the computational fragment of maths: computations on any domain can be encoded as arithmetic computations.

Once we've shown that every recursive function is expressible in L_A, what remains to be shown is that if we code the elements of other domains as numbers, computable operations on these domains will be recursive operations on numbers.

The theory Q as a toy arithmetical theory.

Review: What do models of Q look like? Are all models isomorphic? If so, Q is complete, by completeness of predicate logic!

But Q has nonstandard models. It is incomplete, and that's not hard to see. Gödel's proof of the incompleteness of Q also works for any *stronger* theory, including PA and ZFC, whose incompleteness is much harder to spot.

We can ask what Q knows about the numbers. Does it know that 2+2=4? Yes. It knows all atomic facts about addition and multiplication.

It means that Q can prove all true $_0$ sentences. (And so also all true $_0$ sentences.)

[Representability is only needed for the syntactic incompleteness theorem. I might combine the discussion of Q with the semantic incompleteness theorem into a first chapter? We'll then define two versions of the diagonal lemma in successive chapters, but maybe that's not so bad.]

8.2 Expressibility

The language L_A is very poor. That's deliberate. We can define other concepts in terms of addition, multiplication, successor. That's considered better than having them all primitive.

Compare how we defined $\wedge$ and $\vee$ in propcal.

Can you define exponentiation?

All recursive functions are expressible in the language of arithmetic.

Try to translate $x^2 < y!$. [This could be an exercise, with a warning that it's very very hard.]

x^2 and $<$ are easy. How do we translate $x!$? We'll translate this not into a function expression but into a formula $F(x, y)$ that holds between x and y iff $x! = y$. That's OK, because we can then translate $x^2 < y!$ as, say, $\forall y(F(x, y) \to x^2 < y)$.

The trick is to see the p.r. definition of ! as defining a sequence.

Let fact$_k$ be the sequence $0! \ldots k!$ Our formula $F(x, y)$ will say that y is in kth element of fact$_k$.

Let's pretend, for now, that we variables σ for sequences of numbers, and we have an operation that allows extracting the items of the sequence.

We can define the sequence fact$_k$ as follows: σ is fact$_k$ iff $\sigma[0] = 1 \wedge \forall x < k(\sigma[x+1] = \sigma[x] \cdot (x + 1))$. This is expressible in L_A.

Now we only need to code sequences of numbers into numbers such that we can access their elements. The access function is called β. So if s codes σ then $\beta(s, i) = \sigma[i]$.

8.3 Representability

Remember that Q can prove all particular facts about addition.

This means that x+y *captures* addition.

Remember that + and * are arbitrary symbols. They don't come with a built-in meaning (unlike $\wedge$ and $\neg$). It's really $f_1(x, y)$ that captures x+y.

We've seen that there's a formula $F(x, y)$ that expresses the factorial function, in the sense that $F(n, m)$ is true iff $x! = y$. This connection between F and the factorial function, however, relies on the meaning of the arithmetic expressions 0, +, *, s. A formal theory is just a set of expressions. It doesn't "know" what they mean. So while it is *true* that, for example, $F(3, 6)$, and not that $F(3, 7)$, a formal theory may say the opposite.

We'll be interested in what a theory entails about factorials, or about other recursive notions – in particular, about proofs. To this end, we must find expressions in the language that not merely *express* these functions, but *represent* them. The empty theory, for example, says nothing about factorialhood, because there's no expression that represents it.

We've talked about numerical functions and relations.

Derivatively, an arbitrary set of things is called recursive (or r.e.) if the set of their code numbers is recursive (or r.e.), given some effective method of coding.

8.4 Arithmetization of syntax

Hilbert had the insight that we can study proofs in, say, PA as mathematical objects. What branch of maths do we need? Proof theory. But we can interpret proof theory in set theory. Indeed, Gödel saw that we can interpret it in arithmetic.

We'll code sentences and proofs as numbers. We'll find a p.r. predicate Prf that holds between x and y iff x codes a PA-proof (or Q-proof etc.) of the sentence coded by y.

We're interested in properties and relations on L_A strings. So we need to code these as numbers.

We've talked about coding strings in ch.6. There I suggested that we can code an L_A string in two steps.

First, we assign a code number to each L_A symbol. For example

Symbol	$\neg$	$\rightarrow$	$=$	$\forall$	0	s	$+$	$\times$	(	)	,	x_1	x_2	...
Code	1	2	3	4	5	6	7	8	9	10	11	12	13	...

Then we code formulas, which are sequences of symbols. Code numbers of formulas are called Gödel numbers.

Since we have coded each symbol by a number, we need to code sequences of numbers. We can use the prime factor decomposition technique from the previous section. Take the string '0 = 0'. This will be coded as $2^5 \times 3^3 \times 5^5$, or 2345xxx.

In general, let p_i be the i-th prime. Then the gn of a string $a_1 \dots a_n$ of symbols is

$$p_1^{a_1} \cdot p_2^{a_2} \cdots p_n^{a_n}.$$

(The empty sequence is coded by the empty product, 1.)

Given these two methods, it is possible to code an arbitrary expression of the language by a single number: first, replace each symbol s by it symbol number #(s). This way a sequence of symbols becomes a sequence of numbers. Second, using the above powers of primes coding, associate to this sequence of numbers a unique single number as its code.

The Gödel number of a formula (sentence, derivation) A is denoted by $\ulcorner A \urcorner$. E.g.

$$\langle 0 = 0 \rangle = 2^5 \cdot 3^5 = 243.$$

We can now define arithmetical predicates that indirectly talk about L_A strings. E.g., "the first symbol of the string coded by x is '0'". This holds of a number x if and only if the first exponent is 5, the Gödel number of '0'.

Define:

Var(x) for "x codes a variable symbol in L_A".

Define:

Term(x) for "x codes a term in L_A".

This is p.r.: we need to check if x codes 0 or a variable of a sequence beginning with '$s($' followed by a term followed by '$)$'.

Formula(x) for "x codes a formula in L_A".

This is directly definable by primitive recursion.

Define the substitution function *sub(x,v,t)* that takes a code number x of a formula and a code number v of a variable and t of a term, and returns the code number of the formula obtained by substituting the term for all occurrences of the variable in the formula.

This is tedious but mechanical to construct.

Algorithm: walk through the sequence coded by x; whenever you meet the single-symbol sequence v that is free (check the binding structure on the fly) splice in the code for t. All the operations "read symbol i", "write symbol", "concatenate" and "test bound-/free" are p.r.

A mathematical proof consists of a sequence of formulas. So Gödel gave every sequence of formulas a unique Gödel number too. In this case, he starts with the list of prime numbers as before — 2, 3, 5 and so on. He then raises each prime to the Gödel number of the formula at the same position in the sequence ($2^{243,000,000} \times \ldots$, if $0 = 0$ comes first, for example) and multiplies everything together.

We can now define a p.r. predicate Prf(x,y) that holds between x and y iff x codes a proof of the sentence coded by y from the axioms of Q.

A finite sequence of L_A-strings is a proof if each item satisfies one of the following conditions:

(i) it is an axiom of Q;

(ii) it is a logical axiom;

(iii) it follows from earlier items by MP

(iv) it follows from earlier items by HG

The corresponding predicate Pr(x) is p.r.

(What if we replace Q by another theory? Doesn't matter if the theory is finitely axiomatizable. The proof also works for PA, which is p.r. axiomatized.)

We can define Prf(x,y) as: Pr(x) and Last(x)=y.

This is called the *gödel number* of the string.

We can then define numerical functions that operate on code numbers. For example, we can define a function $len(x)$ that takes returns the length of string coded by a code number x.

We can then define a numerical predicate $sent(x)$ that applies to a number x iff x codes an L_A-sentence.

By a lazy application of Church's Thesis, all these functions are recursive. We've actually shown this for some of them in the previous chapter, where we've shown that they are in fact p.r..

Key propositions about pr functions:

1. Sent(x) is recursive. [BBJ 15.2]

2. Prf(x,y) is recursive. [BBJ 15.3]

Proof sketch

An interesting consequence: the set of sentences deducible from a rec set of axioms is r.e. (BBJ 15.4)

Definition 8.1: Recursive Axiomatizability

A theory is *recursively axiomatizable* if it contains all and only the sentences derivable from some recursive set of axioms. Q and PA and ZFC are recursively axiomatizable.

Exercise 8.1

Show that if a theory is r. ax. and complete then it is decidable. (appeal lazily to Church's Theorem)

9 Incompleteness

See handout wk10.

A theory is *axiomatizable* if it is axiomatized by a decidable set of axioms.

Robinson's Lemma. Every recursive function is numeralwise representable in Q.

9.1 The diagonal lemma

I began with an informal presentation of the diagonal lemma. If we find a sentence that says that it's not provable, we'll get incompleteness. But how can we find such a sentence?

Let's first try to do it in English, without using indexicals. We want to pick out the sentence by its syntactic properties.

Begin with the observation that predicates can be applied to predicates:

- 'is English' is English.
- 'is made of stone' is made of stone.

Call a predicate applied to itself the *diagonalisation* of the predicate. In English, diagonalising a predicate simply means to write it twice next to each other, the first in quotes.

Now consider this predicate:

- 'has a diagonalisation that is not provable'.

Let's diagonalise it:

- 'has a diagonalisation that is not provable' has a diagonalisation that is not provable.

What does this say? It says that the predicate has a diagonalisation. Well every predicate has exactly one. What else does it say about that diagonalisation? That it's not provable. So it says of the diagonalisation of 'has...' that it's not provable. But it *is* that diagonalisation!

Now we do this in the language of arithmetic, using numerals of gns instead of quoted predicates.

9.2 Tarski's Theorem

A proof of Tarski's Theorem that does not mention formal systems at all can be obtained if we replace this step by a use of a semantic form of the Fixed Point Theorem, to wit: for every formula F (y) of L there is a formula H such that F (H) ≡ H is true. Given this, for every formula T (y) of L there will be a sentence H such that ¬T (H) ≡ H is true, and so T (y) fails to be a truth predicate.

To prove the semantic Fixed Point Theorem, we introduce the notion of definability in L. A formula F $(v_1, \ldots, v_m)$ of L defines an m-place relation R just in case, for all $n_1, \ldots, n_m$, F $(n_1, \ldots, n_m)$ is true iff R$(n_1, \ldots, n_m)$. A relation is definable in L iff there exists a formula of L that defines it.

Definability is a semantic correlate of numeralwise representability. Indeed, if F $(v_1, \ldots, v_m)$ numeralwise represents R in a formal system that is sound for the intended interpretation of L, then F $(v_1, \ldots, v_m)$ defines R in L.

The proof of the semantic FPT is just like the proof of the syntactic FPT except that we assume that diag defines, rather than numeralwise represents, diagonalisation. Note that the only properties of L needed are these: the function diag is definable in L; and L contains the usual logical signs

9.3 Church's Theorem

Gödel's work immediately tells us that there is no primitive recursive decision procedure for PA; and that work is extendible to any notion that will be representable in PA. All that was necessary after 1931, then, was to formulate the appropriate general notion of computability, and show that all computable functions were numeralwise representable.

10 The unprovability of consistency

10.1 Fineness of grain

The Montagovian point...

Remember that our models are implausibly coarse-grained. There's more to the meaning of a sentence than its truth-value, or to the meaning of a predicate than its extension.

This doesn't matter if the language doesn't have operators that depend on more than extension.

We could introduce a second-order identity predicate. Then we might want more fine-grained models, so as not to validate the Fregean axiom.

Or we could introduce other non-truth-functional sentence operators. In modal logic, we introduce a box. Models now include world-relative assignments.

How fine-grained the models should be depends on the distinctions we want to draw. We can have operators like Prov that are sensitive to very fine-grained distinctions.

But there are surprising limitations to what operators we can have! …

11 HOL

11.1 Syntax

We can quantify over predicate position. The same intro and elim rules hold for 2nd-order quantifiers.

These are formally easy to define but hard to express in English! Some have thought that this richer language is very useful in mathematics and metaphysics.

Exercise 11.1

Could we quantify over operators or quantifiers?

There's also 2nd-order identity, which is governed by SI and LL (intro and elim).

11.2 Lambdas

A predicate says something about such an object. Informally, if you take a sentence and remove name, leaving a gap, you get a predicate. A predicate isn't true or false – as it contains a gap. Rather it may be true *of* some objects and false *of* others.

[Following Bostock here; later intro lambdas!]

Consider the sentence 'Horace loves Horace'. This could be understood as applying a one-place predicate to Horace, or ....

To mark this difference, we need different gap markers: '– loves –' vs '– loves __'. Instead of gap markers, we can use variables. Thus we can distinguish 'x loves x' from 'x loves y'.

Hmm. But now sentence formation is substitution and F isn't a predicate? Well, yes, F is a predicate letter! Move to lambda chapter. We need complex predicates just as we need complex sentences.

Exercise 11.2

Can you think of a way to construct complex singular terms from formulas? [iota]

Goldfarb sec. 5.4.

11.3 models

We can quantify over predicate position. As a first shot, we could read XXa as Fa Ga ... But what if some properties are not expressed by predicates? I assume that ”property” is the label for the kinds of things expressed by predicates. These plausibly aren’t extensions. Remember that we use extensions to *model* interpretations. Given that sets of individuals model properties, it’s natural to model second-level predicates as ranging over sets of individuals.

Note that this makes properties ”abundant”. We’ve put no constraints on what predicates can mean. For any set of individuals, [F] could be assigned that set. We could put restrictions on the eligible sets, but let’s not do this.

2nd-order logic quantifies into the position of predicates and propositions. We assume that there are ”entities” of the higher type.

We *model* this by assuming that the 2nd-order quantifiers range over arbitrary sets of individuals, just as we modeled the meaning of 2nd-order predicates as arbitrary sets.

11.4 Categoricity

Definition 11.1: Categoricity

Categoricity is the property of a theory that all its models are isomorphic. Explain what this means. For two models $\langle \mathbb{N}; 0, S, +, \cdot \rangle$ there’s a bijection f...

Second-order PA is categorical.

Exercise 11.3

Explain why a theory without = can never be categorical. [You can always duplicate the elements of a model.]

Exercise 11.4

What if you add to PA2 infinitely many statements $a \neq 0, a \neq s(0), ...?$ Is this theory consistent? What do its models look like?

Yes, consistent. No models. https://math.stackexchange.com/questions/1005283/whats-an-example-of-a-theory-thats-consistent-yet-has-no-model

The ”structuralist” approach to mathematics assumes that the study of arithmetic is the study not of particular things – numbers – but of a more abstract structure. The structure can be instantiated in many ways. This structure can only be adequately described in second-order logic.

The same is true for all infinite mathematical structures.

Remember that standard 2nd order logic assumes that the second-order quantifiers range over sets. One might worry that this undercuts the structuralist approach. We’d now need to be assured that the set-theoretic universe is definite.

Hamlin p.32