Advanced Logic

—-

Lecture Notes

Toby Meadows

E-mail address: toby.meadows@gmail.com

I would like to thank Aleksi Anttila, Adam George and Justine Retford for welcome suggestions in making these notes clearer and for their patience in spotting typos in the document.

READ ME:

These notes are still drafts:

- there will be typos;
- there may be errors; and
- I plan to augment them.

That said, they should be complete enough to be useful and I hope you find them so.

I plan to update this document on my website:

https://sites.google.com/site/tobymeadows/.

Unless you've already come from there, it could be worth looking there for a more recent version of this document.

Also, if you do spot any problems or there's something you don't like or understand, I'd like to hear about it. Please drop me an email at:

toby.meadows@gmail.com.

Contents

Part 1

Models & Proofs

CHAPTER 1

Languages and Induction

Goals:

(1) Define the language of first order logic.
(2) Use recursive definitions.
(3) Familiarise ourselves with mathematical proofs.
(4) Use mathematical induction.

1.1. The Language of First Order Logic

1.1.1. The pieces. The language consists of three kinds of symbols:

- logical symbols;
- non-logical symbols; and
- individual variables.

1.1.1.1. *Logical symbols.* The logical symbols are $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \exists, ($ and $)$.

1.1.1.2. *Non-logical symbols.* The non-logical symbols come in three types:

- *constant symbols* - $a, b, c, ...$;
- *relation symbols* - $P, Q, R, ...$; and
- *function symbols* - $f, g, h, ...$

Each of the relation and function symbols have an *arity* which reflects how many names or variables they take as arguments.

We shall often collect together the non-logical vocabulary into a set $\mathcal{L}$:

$$\mathcal{L} = \{a, b, c, ..., P, Q, R, ..., f, g, h, ...\}.$$

1.1.1.3. *Individual variables.* The individual variables are $v_1, v_2, ...$.

We shall denote an arbitrary variables by $x, y, z, ...$

x, y, z are sometimes known as *metavariables*. This is because they aren't strictly part of the language we are studying: the *object language*. They are part of the *metalanguage*.

1.1.2. Terms and formulae

. PROBLEM: We need some way of saying when a string of these symbols is correct or grammatical; then we shall say that it is *well-formed*.

The guiding idea of our definition is as follows:

- We shall take some basic strings which are obviously well-formed. We shall call these *atoms*.
- Then we describe some rules which allows us to take well-formed string and build other well formed string.
- If a string can be eventually constructed out of atoms using the rules, then it is well-formed. Nothing else is a well formed string.

This is called a *recursive definition*.

1.1.2.1. *A warmup definition .* Let's use a simpler language. Let's suppose that:

- the non-logical vocabulary consists of two *propositional symbols*, p and q;
- the logical vocabulary consists of just the symbol $\wedge$.

We might then define a well-formed string as follows:

- Let p and q be atoms; thus they are well-formed string.
- If we have two well formed string, say φ and χ, then $\varphi \wedge \chi$ is a well formed string; i.e., the string formed by placing the $\wedge$ symbol between them is itself a well formed string.
- If φ is a string formed from atoms following the rule, then φ is a well formed string; $\varphi \in WFS$. Nothing else is a well formed string.

REMARK 1. Note the use of the symbols φ and χ to stand for arbitrary strings. Are they part of language?

We can also define WFS by a kind of construction of *stages*:

- at the first stage, we start with p and q (the atoms) which we know are well-formed;
- then at the next stage, we form all of the strings that can be formed from them using the rules - giving us $p \wedge q, q \wedge p, p \wedge p, q \wedge q$;
- we then keep repeating the process; and
- if φ gets into one of the stages, then it is in WFS.

More formally,

DEFINITION 2. $\varphi \in WFS$ iff there is some stage n such that $\varphi \in Stage(n)$ (the n^{th} stage of the construction) where:

- $Stage(1) = \{p, q\}$; and
- $Stage(n+1)$ is the set of φ such that either:
  (1) $\varphi \in Stage(n)$; or
  (2) φ is of the form $\psi \wedge \chi$ where $\psi, \chi \in Stage(n)$.

(1.) just ensures that everything from the previous stages is included in the next; thus, we accumulate everything we built beforehand as we go.

It turns out that these two definitions are equivalent.

1.1.2.2. *First order logic.* Using the same process, we now define the terms and well-formed formulae of first order logic.

Intuitively, a terms is something like a name. We build up the definition of a *term* using a recursive construction:

- (Base) if t is a constant symbol or an individual variable, then t is a term.
- (Rule) if $t_1, ..., t_n$ are terms and f is a function symbol with arity n, then $f(t_1, ..., t_n)$ is a term.

EXAMPLE 3. Let $\mathcal{L}$ have two one function symbol f with arity 2 and a constant symbol a. Then the following are terms:

- a;
- v_1;
- $f(v_1 a)$; and
- $f(v_1, f(v_1, a))$.

Let us call the set of terms $Term$.

A string φ is an *atom* if:

- $\varphi = Rt_1, ..., t_n$ where $t_1, ..., t_n \in Term$ and R is a relation symbol with arity n.

We then set out the process for constructing *well formed formulae* below:

- if φ is an atom then $\varphi \in WFF$;
- if $\varphi \in \{\neg\psi, \forall x\psi, \exists x\psi\}$ where $\psi \in WFF$ and x is a variable, then $\varphi \in WFF$;
- if $\varphi \in \{(\psi \wedge \chi), (\chi \vee \psi), (\psi \rightarrow \chi), (\psi \leftrightarrow \chi)\}$ where $\psi, \chi \in WFF$, $\varphi \in WFF$;

- nothing else is in WFF.

REMARK 4. The logical vocabulary and individual variables will be the same for the remainder of this course. Thus when we speak of a language $\mathcal{L}$ we shall often just mean the non-logical vocabulary. This shouldn't cause too much confusion. Also observe that we are including brackets around each of the formulae before when we compose new formulae from them.

We can also define the same set using our stage based approach.

DEFINITION 5. $\varphi \in WFF$ iff there is some stage n such that $\varphi \in Stage_{WFF}(n)$ (the n^{th} stage of the construction) where:

- $Stage_{WFF}(1)$ is the set of formulae of the form $Rt_1, ..., t_n$ where R is an n-ary relation symbol and $t_1, ..., t_n$ are terms; and
- $Stage_{WFF}(n+1)$ =is the set of φ such that either:
  (1) $\varphi \in Stage_{WFF}(n)$; or
  (2) $\varphi \in \{(\psi \wedge \chi), (\chi \vee \psi), (\psi \rightarrow \chi), (\psi \leftrightarrow \chi), \neg\psi, \forall x\psi, \exists x\psi\}$ where $\psi, \chi \in Stage_{WFF}(n)$ and x is a variable.

1.1.3. Bound variables, free variables and sentences. Intuitively a variable is *free* if it occurs outside the *scope* of all of the quantifiers in the formula. A variable is *bound* if it is not free.

We'll give some examples of this and leave it as an exercise to define the relation x is free in φ using a recursive definition.

EXAMPLE 6. The variable v_1 is free Fv_1.

EXAMPLE 7. The variable v_2 is bound in $\forall v_2 Fv_2$.

EXAMPLE 8. The variable v_1 is bound in $\forall v_1(\exists v_2 Rv_1v_2 \wedge Pv_1)$.

EXAMPLE 9. The variable v_1 is free in $\forall v_2(Rv_1v_2 \wedge \forall v_3(Gv_1 \rightarrow Fv_1))$.

EXAMPLE 10. The variable v_1 is free in $\forall v_1(Pv_1 \rightarrow Qv_1) \wedge Pv_1$. Why?

DEFINITION 11. A *sentence* is a formula with no free variables. We shall denote the sentences of some language $\mathcal{L}$ by $Sent_{\mathcal{L}}$ (omitting the $_{\mathcal{L}}$ where confusion will not arise). An atomic sentence is an atomic formula with no free variables.

1.1.4. A few simplifications in the definition of WFF. Observe that according to our definition of WFF, the following formulae are not well-formed:

(1) $Fa \wedge Gb \wedge Gc$ (our rules say that either $Fa \wedge Gb$ or $Gb \wedge Gc$ should have brackets around it)
(2) $Fa \vee Gb \vee Gc$ (our rules say that either $Fa \vee Gb$ or $Gb \vee Gc$ should have brackets around it)
(3) $\forall x Px$ (This sentence uses a metavariable x which is not actually part of the language.)

Now this clearly a little annoying as we should be able to intuitively see that none of these things would make any difference to the meaning of the formula in question (although we haven't really talked about meaning yet).

Moreover, it can make things very difficult to read. Consider

$$\forall v_1(\neg Rv_1v_2 \rightarrow \exists v_3(Hv_3v_1 \wedge (\exists v_2 \neg Gv_1v_2 \wedge Pv_3)))$$

as opposed to

$$\forall x(\neg Rxy \rightarrow \exists z(Hzx \wedge \exists y \neg Gxy \wedge Px)).$$

Thus, I'm often going to use a *sloppier* version of WFF, which allows us to write formulae in the simpler form above.

In situations where is important to use the *strict* definition I will endeavour to explicitly mention this. You should try to see that the translation back to the strict language is very trivial and as such, the difference will *rarely* matter.

1.2. Induction

Mathematical induction is, in a sense, the other side of coin for recursive constructions we've considered in the previous section.

1.2.1. Motivation. Consider the natural numbers $\mathbb{N} = 0, 1, ...$ and consider some subset $A \subseteq \mathbb{N}$; i.e., a collection of natural numbers.

Now suppose that $0 \in A$.

Moreover, suppose that for any natural number n, if $n \in A$, then $n + 1 \in A$. So given any number, if it's in A, then so is the next one.

If this is the case, then clearly every number n must be in A.

FACT 12. *(IND) Suppose* $A \subseteq \mathbb{N}$. *If* $0 \in A$ *and* $\forall n(n \in A \rightarrow n+1 \in A)$, *then* $\forall n\ n \in A$.

1.2.1.1. *The least number principle.* If the previous principle isn't obvious, the perhaps this one is.

FACT 13. *(LNP) If* $B \subseteq \mathbb{N}$ *is not empty, then there is some* $n \in B$ *such that for all* $m < n$, $m \notin B$; i.e., there is some least member of B.

Of course, the least member could be 0.

Let's prove IND, using LNP.

THEOREM 14. *LNP* $\rightarrow$ *IND.*

REMARK 15. We are going to produce a *reductio ad absurdum*. Intuitively, the idea is that we'll take the LNP as given and then see what happens if we assume that IND is not true. We'll show that this leads us to contradict ourselves: and *absurdity*. Thus, our assumption that IND is not true, is itself wrong. Thus IND is actually correct.

PROOF. Assume LNP. Now suppose for *reductio*, that IND is not correct. Then there is some $A \subseteq \mathbb{N}$, such that:

(1) $0 \in A$;
(2) $\forall n(n \in A \rightarrow n+1 \in A)$; but
(3) there is some n, such that $n \notin A$.

Let $B = \mathbb{N} \backslash A$; i.e., the set of natural numbers that are not in A. This not empty by (3.) By LNP, there is some least member of B. Let's call it b. Now b is a natural number so either:

- $b = 0$; or
- $b = n + 1$ for some n.

In the first case, this would mean $b = 0 \in B$ and thus $0 \notin A$, contradicting (1.). In the second case, since $b = n + 1$ is the least member of B, we have $n \in A$, but $b = n + 1 \notin A$, contradicting (2.).

This tells us that there cannot be such a B; i.e., that for all n, we actually have $n \in A$. Thus IND is correct. □

Indeed, we can actually go the other way too. We can prove LNP from IND.

THEOREM 16. *IND* $\rightarrow$ *LNP.*

PROOF. Suppose IND is true, but for a *reductio*, suppose that LNP is not. Then there is some subset $B \subseteq \mathbb{N}$ such that:

(1) B is not empty; and
(2) B has no least element.

Let $A = \mathbb{N} \backslash B$. Clearly $0 \notin B$ (for then 0 would be the least element), so $0 \in A$. Moreover, since B has no least element, this just means that for any $n \in \mathbb{N}$, if $n+1 \in B$, then $n \in B$; otherwise, such an n would be the least. But this means that for any n if $n+1 \notin A$, then $n \notin A$; and by contraposition, if $n \in A$, then $n+1 \in A$. Thus we have both the antecedent conditions of IND and so for all n, $n \in A$; i.e., $n \notin B$. But this means B is empty contradicting (1.). Thus LNP must be true after all.

□

REMARK 17. So there is a sense in which they actually both mean the same thing.

1.2.2. Using induction.

1.2.2.1. *A simple mathematical proof.* A *triangular number* n is such that

$$n = m + (m-1) + ... + 2 + 1$$

for some m.

We can use induction to prove the following fact about natural numbers.

THEOREM 18. *For all* m, $m + (m-1) + ... + 2 + 1 = \frac{m(m+1)}{2}$.

REMARK 19. This a universal statement that we want to prove by induction. We are going to let A be the set of numbers m such that

$$m + (m-1) + ... + 2 + 1 = \frac{m(m+1)}{2}.$$

That is, every $m \in A$ is such that the equation above holds of m.

To use induction, we need to show that:

(1) $0 \in A$; and
(2) $\forall n$, if $n \in A$, then $n+1 \in A$.

From there, we can conclude that every n is in A.

We shall call step (1.) the *base case*; and step (2.) the *induction step*.

PROOF. By induction.

(Base) We need to show that $0 \in A$. For this to be true we just need

$$\begin{aligned} 0 &= \frac{0(0+1)}{2} \\ &= 0/2 \\ &= 0 \end{aligned}$$

which is obviously correct.

(Induction Step) We take some $n \in A$ and show that $n+1 \in A$. The fact that $n \in A$ just means that

$$n + (n-1) + ... + 2 + 1 = \frac{n(n+1)}{2}.$$

Then consider the next triangular number

$$(n+1) + n + (n-1) + ... + 2 + 1.$$

Then with a little bit of simple maths, we get

$$\begin{aligned} (n+1) + n + (n-1) + ... + 2 + 1 &= (n+1) + \frac{n(n+1)}{2} \\ &= (n+1) + \frac{n^2+n}{2} \\ &= \frac{(2n+2) + (n^2+n)}{2} \\ &= \frac{n^2+3n+2}{2} \\ &= \frac{(n+1)(n+2)}{2} \\ &= \frac{(n+1)((n+1)+1)}{2} \end{aligned}$$

But this just means that $n+1 \in A$.

Thus by IND, we see that every n is in A, which is what we wanted to show. □

1.2.2.2. *Using induction on a language.* Let's go back to the toy language from Section 1.1.2.1.

It should be obvious that any $\varphi \in WFS$ (i.e., a well formed string) is either atomic or of the form $\chi \wedge \psi$ for some $\chi, \psi \in WFS$.

But how would we *prove* this? We use induction.

THEOREM 20. *For every $\varphi \in WFS$ either φ is atomic or φ is of the form $(\psi \wedge \chi)$ for some $\chi, \psi \in WFS$.*

But how to do we use induction here? These strings are *not* natural numbers.

Consider Definition 2, which used stages *construct* all of the well formed strings. Each stage of the definition was *indexed* by a particular natural number n.

We shall use the *stage indices* to make a proof by by induction. Thus we shall demonstrate that the proposition holds for every stage n of the construction and thus for everything in WFS.

Thus we shall make A the set of n such that $\varphi \in Stage(n)$ iff either:

- φ is atomic; or
- φ is of the form $\psi \wedge \chi$ where $\psi \wedge \chi \in Stage(m)$ for some $m < n$.

PROOF. By induction on the stage of the construction of WFS.

(BASE) We need to show that $0 \in A$, which just means that for all $\varphi \in Stage(0)$ either φ is atomic or φ is of the form $\psi \wedge \chi$ where $\psi \wedge \chi \in Stage(m)$ for some $m < 0$. There are no stages $m < 0$, so we don't have to worry about the second part. Moreover, since $Stage(0) = \{p, q\}$, it is obvious that everything in $Stage(0)$ is atomic which is sufficient.

(INDUCTION STEP) Suppose $n \in A$. We must show that $n+1 \in A$. Since $n \in A$, every $\varphi \in Stage(n)$ is either atomic or of the form $\psi \wedge \chi$ for $\psi, \chi \in Stage(m)$ for some $m < n$. Thus we only need to worry about those strings added at stage $n + 1$. Suppose $\varphi \in Stage(n + 1) \backslash Stage(n)$; i.e., it's one of the new things added at stage $n+1$. Then by the way we defined the rule of construction for $Stage(n+1)$, φ must be of the form $\psi \wedge \chi$ for some $\psi, \chi \in Stage(n)$. So $n+1 \in A$.

Then by induction we see that every $n \in A$. Thus for all $\varphi \in WFS$, φ is either atomic or φ is of the form $\psi \wedge \chi$ for some $\psi, \chi \in WFS$. □

Now this is a very simple proof. In a sense, we used the definition by recursion to construct a set of well formed strings and then we used induction to show that they indeed were well formed.

We'll usually, however, be concerned with:

- a more complex language (first order logic); and
- a more difficult problem.

As such, we often want a way of *streamlining* the kind of proof we did above. Rather than concerning ourselves with stages in a construction, we often define the *complexity* of a formula.

DEFINITION 21. Let $\varphi \in WFF$ be a well formed formula of the language of first order logic. We define the complexity of φ, abbreviated $comp(\varphi)$, by recursion as follows:

- if φ is atomic, then $comp(\varphi) = 0$;
- if $\varphi \in \{\neg\psi, \forall x\psi, \exists x\psi\}$ and $\psi \in WFF$, then $comp(\varphi) = comp(\psi) + 1$;
- if $\varphi \in \{(\psi \wedge \chi), (\psi \vee \chi), (\psi \rightarrow \chi), (\psi \leftrightarrow \chi)\}$ and $\psi, \chi \in WFF$, then

$$comp(\varphi) = max(comp(\psi), comp(\chi)) + 1,$$

where $max(m, n)$ is the maximum of m and n.

Now suppose we want to construct a proof of some fact about all the well formed formulae. If we want to use induction, we may do this by doing the induction on the complexity of the formulae.

REMARK 22. The complexity amounts to much the same thing as the stage in the construction.

It is conventional in proofs by induction on complexity not to explicitly mention the complexity of the formulae. The reason for this is that now we understand the principles underpinning the induction, we don't really need to think about the numbers anymore.

When we make a proof by induction on the complexity/stage of formulae, we really just need the following steps:

(1) (BASE) show that the property holds for the atomic case - this just the 0-level of complexity.
(2) (INDUCTION STEP) show that if the property holds for two well-formed formulae, then it holds for any well formed formula that can be constructed by the rules from them - this is just showing that we can move from any level of complexity to the next.

In the case of first order logic, verifying the induction step entails verifying that new formulae constructed using the rules still have the desired property.

Exercises Week 1

EXERCISE 23. What change in the rules (of the recursive definition) would you make to WFF in order to formalise the sloppy rules of WFF in such a way to accommodate the kind of problem illustrated in (1.) and (2.) of 1.1.4?

EXERCISE 24. In which of the following formulae is v_1 bound:

(1) $\forall v_1 P v_1 v_2$;
(2) $\forall v_2(Pv_1v_2 \wedge \exists v_3 R v_1 v_2)$;
(3) $\forall v_2(Pv_2 \wedge \exists v_1 R v_1 v_2)$;
(4) $\forall v_2(Pv_2 \wedge (\exists v_1 R v_1 v_2 \rightarrow Pv_1))$.

EXERCISE 25. Are all formulae sentences? Are all sentences formulae?

EXERCISE 26. Which of the following formulae are sentences:

(1) $\forall v_1 P v_1 v_2$;
(2) $\forall v_2(\exists v_1 P v_1 \wedge \forall v_2 R v_1 v_2)$;
(3) $\forall v_2(Pv_1 \rightarrow \exists v_1 R v_1 v_2)$;
(4) $\forall v_1(Pv_1 \rightarrow \exists v_2 R v_1 v_2)$.

EXERCISE 27. Develop a recursive definition for the relation:

- the variable x is free in the formula φ.

[Try to do this in stages. Define the *atomic case* and then *rules* for getting from one stage to the next. Given an atomic sentence of the form $\varphi := Pt_1, ., t_k, .., t_n$ where $x = t_k$ is a variable (and a term) we can say that x is free in $Pt_1, ..., t_n$.]

EXERCISE 28. Prove (using induction) that every string of our toy language WFS contains an odd number of symbols. [Clearly the atoms contain an odd number of symbols. You need to construct the induction hypothesis that will allows to prove the claim for every level of complexity.]

CHAPTER 2

Models

Goals:

(1) Define a model.
(2) Describe what it means for a sentence to be true in a model.
(3) To define some important metatheoretic properties: including validity and satisfiability.

By what of introduction, we might think of a model as something for one of our languages to *talk about*. It will consist of things which bear certain relations to other things in the model. Moreover, we shall be able to use sentences in the language to *talk about* the model: to express things about what is going on inside the model.

2.1. What is a model?

Intuitively speaking, a model is an interpretation of a language $\mathcal{L}$. By language, we simply mean the non-logical vocabulary of constant, relation and function symbols. It tells us what the language means.

Thus a model $\mathcal{M}$ has three *ingredients*:

(1) A language $\mathcal{L} = \{a, b, c, ..., P, Q, R, ..., f, g, h\}$;
(2) A domain M of objects - which we can then talk about using the language.
(3) An interpretation of the language such that:
   - Constant symbols like c are interpreted by some object from the domain $c^{\mathcal{M}} \in M$ - this the object which the name denotes.
   - Relation symbols like R where the arity of R is n are interpreted by some set of tuples of objects $\langle m_1, ..., m_n \rangle$ where $m_1, ..., m_n \in M$. Thus the interpretation of R in $\mathcal{M}$, abbreviated $R^{\mathcal{M}}$, is a subset of $\{\langle m_1, ..., m_n \rangle \mid m_1, ..., m_n \in M\}$.
   - Function symbols like f with arity n are interpreted by functions taking some $m_1, ..., m_n$ from $\mathcal{M}$ and returning some $m \in M$.

To get the idea of the interpretation of a relation symbol, consider a one place relation symbol P. The interpretation of P in $\mathcal{M}$, $P^{\mathcal{M}}$, is simply the set of objects from the domain M which are in the extension of the P according to $\mathcal{M}$.

A model $\mathcal{M}$ of $\mathcal{L} = \{a, b, c, ..., P, Q, R, ..., f, g, h\}$ is thus something of the form

$$\mathcal{M} = \langle M, a^{\mathcal{M}}, b^{\mathcal{M}}, c^{\mathcal{M}}, ..., P^{\mathcal{M}}, Q^{\mathcal{M}}, R^{\mathcal{M}}, ..., f^{\mathcal{M}}, g^{\mathcal{M}}, h^{\mathcal{M}} \rangle.$$

REMARK 29. A tuple is simply a finite list of objects. We represent a tuple of $m_1, ..., m_n$ by $\langle m_1, ..., m_n \rangle$.

2.1.1. Examples.

EXAMPLE 30. Consider a language $\mathcal{L} = \{a, P, R\}$ consisting of: a constant symbol a; a one place relation symbol P; and a two place relation symbol R (i.e., they have arity 1 and 2 respectively). Let us define a model $\mathcal{M}$ for $\mathcal{L}$.

- Let the domain M consist of two objects m_1 and m_2.
- Let $a^{\mathcal{M}}$ (the interpretation of a in $\mathcal{M}$) be the object m_1 from the domain M.
- Let $P^{\mathcal{M}}$ (the interpretation of the symbol P in $\mathcal{M}$) be the set $\{m_1\}$.
- Let $R^{\mathcal{M}}$ be the set $\{\langle m_1, m_1 \rangle, \langle m_1, m_2 \rangle\}$.

We have given the language an domain and interpretation, so $\mathcal{M} = \langle M, P^{\mathcal{M}}, R^{\mathcal{M}} \rangle$. Intuitively speaking, the fact that $\langle m_1, m_2 \rangle \in R^{\mathcal{M}}$ tells us that m_1 bears the relation $R^{\mathcal{M}}$ to m_2. In the next section we are going to show how to use sentences of the language to express this fact.

REMARK 31. Note that $R^{\mathcal{M}}$ is a set of tuples from the domain, but R is just a (relation) symbol from the language.

EXAMPLE 32. Imagine a room full of people: John, Mary, Peter and Dorothy. We are going to let them from the domain of a model $\mathcal{M}$. Let us have a language consisting:

- a one place relation B which we shall interpret as the set of boys;
- a one place relation G which we shall interpret as the set of girls;
- a two place relation D which is interpreted as all the pairs $\langle m_1, m_2 \rangle$ (tuples of length 2) such that m_1 danced with m_2.

This defines a model.

REMARK 33. In the example above, I have used B because it starts with the same letter as the word "boys", which makes things easier to remember. However, there is nothing stopping me from defining a different interpretation (and thus different model) in which I made B *denote* (be interpreted as) the set of girls and G denote the set of boys; or let B denote all of the object in the domain.

EXAMPLE 34. Consider the language of arithmetic

$$\mathcal{L}_{Ar} = \{0, 1, +, \times\}.$$

0 and 1 are constant symbols and $+$ and $\times$ are function symbols. The *standard model* of arithmetic consists of a domain N which consists of all of the natural numbers and interprets:

- the constant symbol 0 as denoting the number zero;
- the constant symbol 1 as denoting 1, the next number;
- interprets the function symbol $+$ as addition; and
- interprets the function symbol $\times$ as denoting multiplication.

We shall denote this model as $\mathbb{N} = \langle N, 0^{\mathbb{N}}, 1^{\mathbb{N}}, +^{\mathbb{N}}, \times^{\mathbb{N}} \rangle$.

REMARK 35. We can think of a model as being something like a *possible world*. The domain consists of all of the objects in that possible world and then we interpret the language according to what is true in that world.

However, there are limitations to the analogy. Discussion of possible worlds tends to be limited to an entire physical reality. Models, however, can be based on just a few objects from this world. Moreover, the object in a model might not be concrete: they could be abstract and they might not even need to exist at all.

Also, while we can say things about a possible world using a language, possible worlds are supposed to be, in some sense, independent of the language we use to describe them; a model is not. The language gives the structure to the model.

2.2. Satisfaction

In this section, our goal is to find a way of connecting the sentences of our language to a model $\mathcal{M}$ of that language $\mathcal{L}$. We want to use our language

to express facts about what is going on the model. Thus we want to know when some sentence is true in $\mathcal{M}$.

If a sentence φ is true in $\mathcal{M}$, we shall say that $\mathcal{M}$ satisfies φ, we we abbreviate $\mathcal{M} \models \varphi$.

We shall do this in stages:

(1) we show how to see when atomic sentences are true in $\mathcal{M}$;
(2) we show how to use the simple connectives like $\neg$ and $\wedge$;
(3) we encounter a problem with the quantifiers; and
(4) we solve that problem and give our final definition of satisfaction.

2.2.1. Atomic satisfaction. Let $\mathcal{L} = \{a, b, c, ..., P, Q, R, ..., f, g, h\}$ be our general language consisting of constant, relation and function symbols. We shall, however, ignore the function symbols for the moment.

Let φ be an atomic sentence. Then is must be of the form $Pa_1...a_n$ where P is a n-ary relation symbol and $a_1, ..., a_n$ are constant symbols from $\mathcal{L}$. Since φ is a sentence it cannot have any free variables.

Now let

$$\mathcal{M} = \langle M, a^{\mathcal{M}}, b^{\mathcal{M}}, c^{\mathcal{M}}, ..., P^{\mathcal{M}}, Q^{\mathcal{M}}, R^{\mathcal{M}}, ..., f^{\mathcal{M}}, g^{\mathcal{M}}, h^{\mathcal{M}} \rangle$$

be a model of $\mathcal{L}$.

We would like to know when the atomic sentence φ is true in $\mathcal{M}$; i.e., $\mathcal{M} \models \varphi$.

This is quite simple. We shall say that:

- $\mathcal{M} \models Pa_1...a_n$ iff $\langle a_1^{\mathcal{M}}, ..., a_n^{\mathcal{M}} \rangle \in P^{\mathcal{M}}$.

Intuitively speaking we are saying that $Pa_1...a_n$ is true in $\mathcal{M}$ iff the tuple $\langle a_1^{\mathcal{M}}, ..., a_n^{\mathcal{M}} \rangle$ consisting of the interpretations $a_1^{\mathcal{M}}, ..., a_n^{\mathcal{M}}$ of the constant symbols $a_1, ..., a_n$ is in the interpretation $P^{\mathcal{M}}$ of the relation symbol P.

EXAMPLE 36. Consider the language and model from Example 30 and the sentence Raa. We then have:

$$\begin{aligned} \mathcal{M} \models Raa &\Leftrightarrow \langle a^{\mathcal{M}}, a^{\mathcal{M}} \rangle \in R^{\mathcal{M}} \\ &\Leftrightarrow \langle m_1, m_1 \rangle \in \{\langle m_1, m_1 \rangle, \langle m_1, m_2 \rangle\}. \end{aligned}$$

The first $\Leftrightarrow$ is obtained by our atomic satisfaction definition and the second $\Leftrightarrow$ is obtained by the definition of the interpretations of a and R in $\mathcal{M}$. Since the last statement is clearly correct, we see that Raa is indeed true in $\mathcal{M}$.

2.2.2. Simple connectives. Atomic satisfaction is pretty straightforward. We now want to consider what to do with the connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. To make things simpler, we'll just play with $\neg$ and $\wedge$.

2.2.2.1. *Negation ($\neg$).* Suppose we have some sentence $\varphi \in Sent_{\mathcal{L}}$ which is of the form $\neg\psi$. Then, from last week, we know that ψ is also in $Sent_{\mathcal{L}}$.

Taking an (informal) *inductive* approach, suppose that we already know whether or not ψ is true in $\mathcal{M}$; i.e., whether $\mathcal{M} \models \psi$. Given this knowledge, what can we then say about φ in $\mathcal{M}$?

Suppose it is the case that $\mathcal{M} \models \psi$. Then clearly $\neg\psi$ cannot be true in $\mathcal{M}$ too: this is just what negation means. Thus $\mathcal{M} \not\models \neg\psi$; i.e., $\mathcal{M} \not\models \varphi$. On the other hand suppose we knew that $\mathcal{M} \not\models \psi$. Then, it must be the case that $\mathcal{M} \models \neg\psi$; i.e., $\mathcal{M} \models \varphi$. Again, this is just what negation means: if it is not the case that ψ is true, then ψ is not true (i.e., $\neg\psi$ is true).

With this in mind, we might then say that if φ is of the form $\neg\psi$ then:

- $\mathcal{M} \models \varphi$ iff $\mathcal{M} \not\models \psi$.

REMARK 37. Note that we write $\mathcal{M} \not\models \varphi$ to mean that it is not the case that $\mathcal{M} \models \varphi$.

EXAMPLE 38. Using the language and model of Example 30 again, consider the sentences $\neg Pa$ and $\neg\neg Raa$. For $\neg Pa$ we have the following:

$$\begin{aligned} \mathcal{M} \models \neg Pa &\Leftrightarrow \mathcal{M} \not\models Pa \\ &\Leftrightarrow a^{\mathcal{M}} \notin P^{\mathcal{M}} \\ &\Leftrightarrow m_1 \notin \{m_1, m_2\}. \end{aligned}$$

The first $\Leftrightarrow$ is obtained the clause we just provided for negation; the second $\Leftrightarrow$ is given by the atomic satisfaction clause; and the final $\Leftrightarrow$ is given by the interpretation of the symbols from $\mathcal{L}$ in $\mathcal{M}$.

Since the final clause above is clearly false (it is the case that $m_1 \in \{m_1, m_2\}$) we see that it is not the case that $\mathcal{M} \models \neg Pa$; (i.e., $\mathcal{M} \not\models \neg Pa$).

For $\neg\neg Raa$, we have:

$$\begin{aligned} \mathcal{M} \models \neg\neg Raa &\Leftrightarrow \mathcal{M} \not\models \neg Raa \\ &\Leftrightarrow \mathcal{M} \models Raa. \end{aligned}$$

We established that final clause was true in 36, so we have $\mathcal{M} \models \neg\neg Raa$.

2.2.2.2. *Conjunction ($\wedge$).* Now suppose we had a sentence $\varphi \in Sent_{\mathcal{L}}$ which was of the form $\psi \wedge \chi$ (where, of course, $\psi, \chi \in Sent_{\mathcal{L}}$).

Thinking in an (informally) *inductive* manner, suppose we already knew whether or not it was the case that both of $\mathcal{M} \models \psi$ and $\mathcal{M} \models \chi$ are correct.

Suppose we have $\mathcal{M} \models \psi$ and $\mathcal{M} \models \chi$. Then conjunction just means that $\psi \wedge \chi$ must also be tree in $\mathcal{M}$; i.e., $\mathcal{M} \models \psi \wedge \chi$.

On the other hand, suppose we have either $\mathcal{M} \not\models \psi$ or $\mathcal{M} \not\models \chi$. This just the same as saying that it isn't the case that both $\mathcal{M} \models \psi$ and $\mathcal{M} \models \chi$. Then clearly (since we know what "and" means) it cannot be the case that $\mathcal{M} \models \psi \wedge \chi$; thus we have $\mathcal{M} \not\models \psi \wedge \chi$.

This suggests that we should say the following about conjunction. If $\varphi \in Sent_{\mathcal{L}}$ is of the form $\psi \wedge \chi$ then:

- $\mathcal{M} \models \varphi$ iff $\mathcal{M} \models \psi$ and $\mathcal{M} \models \chi$.

EXAMPLE 39. Returning the model and language of 30, consider the sentence $Raa \wedge Pa$. We have

$$\begin{aligned} \mathcal{M} \models Raa \wedge Pa &\Leftrightarrow \mathcal{M} \models Raa \ \& \ \mathcal{M} \models Pa \\ &\Leftrightarrow \langle a^{\mathcal{M}}, a^{\mathcal{M}} \rangle \in R^{\mathcal{M}} \ \& \ a^{\mathcal{M}} \in P^{\mathcal{M}} \\ &\Leftrightarrow \langle m_1, m_1 \rangle \in \{\langle m_1, m_1 \rangle, \langle m_1, m_2 \rangle\} \ \& \ m_1 \in \{m_1, m_2\}. \end{aligned}$$

Since the final clause is clearly correct, we see that we do have $\mathcal{M} \models Raa \wedge Pa$.

2.2.2.3. *Putting it together.* Using the usual ways of defining connectives, we can then define *satisfaction* using a recursive definition for the connectives (and not the quantifiers).

To make things faster, I'm going to write $\varphi := \ldots$ to mean that φ is of the form $\ldots$.

Suppose $\varphi \in Sent_{\mathcal{L}}$ and that $\mathcal{M}$ is a model of $\mathcal{L}$. Then

- if $\varphi := Ra_1 \ldots a_n$ for R an n-ary relation symbol from R and $a_1, \ldots, a_n$ constant symbols from r, then
  - $\mathcal{M} \models \varphi$ iff $\langle a_1^{\mathcal{M}}, \ldots, a_n^{\mathcal{M}} \rangle \in R^{\mathcal{M}}$;
- if $\varphi := \neg\psi$, then
  - $\mathcal{M} \models \varphi$ iff $\mathcal{M} \not\models \psi$;
- if $\varphi := \psi \wedge \chi$, then

  - $\mathcal{M} \models \varphi$ iff $\mathcal{M} \models \psi$ and $\mathcal{M} \models \chi$;
- if $\varphi := \psi \vee \chi$, then
  - $\mathcal{M} \models \varphi$ iff $\mathcal{M} \models \psi$ or $\mathcal{M} \models \chi$;
- if $\varphi := \psi \to \chi$, then
  - if $\mathcal{M} \models \psi$, then $\mathcal{M} \models \chi$;
- if $\varphi := \psi \leftrightarrow \chi$, then
  - $\mathcal{M} \models \varphi$ iff ($\mathcal{M} \models \psi$ iff $\mathcal{M} \models \chi$)

REMARK 40. Note that in the last clause we needed to resort to brackets. This is one of the drawbacks of using the natural language, English, as the language of the metatheory.

2.2.3. A problem for quantifiers. Observe that the definition above says nothing about what we should do when we encounter a sentence $\varphi \in Sent_{\mathcal{L}}$ which involves quantification.

EXAMPLE 41. Using Example 30 again, consider the sentence $\exists x \neg Px$. Is this correct?

To make things clearer we might represent the interpretation of R using a table as follows (read anti-clockwise):

$P^{\mathcal{M}}$	
m_1	1
m_2	0

A 1 in the box for m_1 represents the fact that $m_1 \in P^{\mathcal{M}}$; and a 0 in the box for m_2 represents the fact that $m_2 \notin P^{\mathcal{M}}$.

Now, intuitively speaking, $\exists x \neg Px$ says (of the model $\mathcal{M}$) that there is some object x such that it is not the case that x in $P^{\mathcal{M}}$. This is clearly true: m_2 suffices since $m_2 \notin P^{\mathcal{M}}$.

2.2.3.1. *The problem.* However, the rules we gave in the previous section don't tell us how to do this. They only say what to do when we are given a sentence composed from atomic sentences using $\neg, \wedge, \vee, \to$ and $\leftrightarrow$.

So clearly we need to say more about how the satisfaction relation works. We need to say how it deals with sentences involving quantifiers.

But there is a further *problem*. We'll try to extend our characterisation of satisfaction in an obvious way and the problem will emerge.

Let's consider our sentence $\exists x \neg Px$ again. We want to be able to say something like this.

- $\mathcal{M} \models \exists x \neg Px$ iff there is some object $m \in M$ such that $m \notin P^{\mathcal{M}}$.

We can see that the latter part is true and thus that $\exists x \neg Px$ is true in $\mathcal{M}$.

But to do this we've had to make a lot of steps at once. In the case of the connectives we were, so to speak, able to unravel one connective at a time as we worked our way to facts about the model itself. However, here we've jumped from a complex sentence involving quantification straight to the model.

While we can see how this works with a relatively simple sentence like $\exists x \neg Px$, it would be much more difficult with a sentence like:

$$\forall y(Ray \rightarrow \exists x \forall z(Pz \rightarrow (Rxz \wedge Rxy))).$$

We want some way of unraveling a sentence one step at a time. The obvious thing to do is to consider how the sentence was built up from formulae of lower complexity. Thus, as we saw in the previous week, $\exists x \neg Px$ was constructed in the third stage.

- Px is atomic and thus there at the beginning.
- $\neg Px$ is constructed by applying $\neg$ to Px and thus gets into the next stage.
- And $\exists x \neg Px$ gets is constructed by applying $\exists x$ to $\neg Px$ and thus gets into the stage after that.

If we follow the pattern of construction of a sentence from the subformulae which are used to construct it, then we'll have the sort of thing we want.

Indeed this is how the definition works for the connectives. For example, we figure out whether $\mathcal{M} \models \psi \wedge \chi$ by checking whether the sentences ψ and χ are true in $\mathcal{M}$.

But here's the problem: if we take pull a quantifier off a sentence, we (may) end up with a formula with a free variable. For example, if we take the $\exists x$ from the front of $\exists x \neg Px$, we end up up with $\neg Px$. This has one free variable (i.e., x) and thus it is not a sentence.

Our characterisation of satisfaction only works for sentences.

This might help illustrate the problem. Say we try to make a rule that *unravels* sentences of the form $\exists x\psi(x)$. We might say something like, if $\varphi := \exists x\psi(x)$ then

- $\mathcal{M} \models \varphi$ iff if [SOME CONDITION HOLDS], then $\mathcal{M} \models \psi(x)$.

For the moment, don't worry about what the condition is, the problem is the statement $\mathcal{M} \models \psi(x)$. What could it mean to say that $\mathcal{M} \models \neg Px$? Intuitively, we are supposed to say something like $\neg Px$ is true in $\mathcal{M}$. But x is just a variable, it doesn't denote anything in the domain. It's meaningless to say that $\neg Px$ is true in $\mathcal{M}$.

2.2.4. Our solution and final satisfaction definition. There are a number of ways of getting around this. We'll consider a simple technique for solving the problem and then make some remarks about other approaches which are common in the literature. However, there is a sense in which they all come down to much the same thing in the end.

2.2.4.1. *Something that doesn't work but points the way to a solution.* So our problem is that we don't have a way of talking about the truth of a formula and this makes it difficult to unravel a complex sentence into simpler parts so that we can figure out whether or not it is true.

So let's not try to give satisfaction conditions for formulae, let's stick with sentences. So if we have a sentence of the form $\exists x\varphi(x)$ we might consider what would happen if we substituted constant symbols from $\mathcal{L}$ in for x. An example might help here:

EXAMPLE 42. Let $\mathcal{M}$ for a language $\mathcal{L} = \{a, b, P\}$ be such that :

- the domain $M = \{m_1, m_2\}$;
- $a^{\mathcal{M}} = m_1$, $b^{\mathcal{M}} = m_2$; and
- $P^{\mathcal{M}} = \{m_1\}$.

This clearly suffices for the definition of a model.

Now suppose we want to know whether or not $\mathcal{M} \models \exists xPx$. To ascertain whether or not this the case, we might ask whether there is some constant symbol c from $\mathcal{L}$ such that $\mathcal{M} \models Pc$.

Clearly, we have $\mathcal{M} \models Pa$ since

$$a^{\mathcal{M}} = m_1 \in \{m_1\} = P^{\mathcal{M}}.$$

Thus we can see that $\mathcal{M} \models \exists x Px$.

We might then be tempted to generalise this and make the following attempt at a rule. If φ is of the form $\exists x \psi(x)$ then

- $\mathcal{M} \models \varphi$ iff for every constant symbol c from $\mathcal{L}$ we have $\mathcal{M} \models \psi(c \mapsto x)$

where $\psi(c \mapsto x)$ means that c has been substituted for x in all of the places where x was free in ψ.

However, it is easy to see that this approach merely gives us a sufficient condition for $\mathcal{M} \models \exists x \psi$. If there is such a constant symbol, then it clearly works. However, there may be situation where we do not have enough constant symbols in the language (perhaps we even have none). An example may illustrate this:

EXAMPLE 43. Let's use the model and language from Example 30 and consider the sentence $\exists x \neg Px$. Then using the proposed quantifier clause we get:

$$\begin{aligned} \mathcal{M} \models \exists x \neg Px &\Leftrightarrow \text{there is some cons-sym } c \text{ such that } \mathcal{M} \models \neg Pc \\ &\Leftrightarrow \text{there is some cons-sym } c \text{ such that } \mathcal{M} \not\models Pc \\ &\Leftrightarrow \text{there is some cons-sym } c \text{ such that } c^{\mathcal{M}} \notin P^{\mathcal{M}}. \end{aligned}$$

However since a is the only constant symbol in $\mathcal{L}$ and $a^{\mathcal{M}} \in P^{\mathcal{M}}$, the last clause is false; and thus, we are told that $\mathcal{M} \not\models \exists x \neg Px$, which, as we know from Example 41, is not correct.

So this doesn't work, since we might not have enough constant symbols available.

2.2.4.2. *Our solution.* But this points the way to an obvious fix. We are going to add more constant symbols to the language.

Given some model $\mathcal{M}$, we need a collection of new constant symbols which provide a name for every object from the domain M.

What should we use for those constant symbols? Simple! We'll just use the objects $m \in M$ themselves.

Given a language $\mathcal{L}$ and model $\mathcal{M}$, we denote the *expansion* of $\mathcal{L}$ with the new constant symbols by $\mathcal{L}(M)$. We can then define $WFF_{\mathcal{L}(M)}$ and $Sent_{\mathcal{L}(M)}$ in the same way we did last week.

Observe that $WFF_{\mathcal{L}} \subseteq WFF_{\mathcal{L}(M)}$ and $Sent_{\mathcal{L}} \subseteq Sent_{\mathcal{L}(M)}$.

REMARK 44. Observe that this is a strange kind of language. Suppose we had started with a model $\mathcal{M}$ for a language consisting just one predicate P and a domain with a single object, me; i.e., $M = \{\text{Toby}\}$. Then I am a constant symbol in the language $\mathcal{L}(M)$ and P "applied to" me is a sentence of $\mathcal{L}(M)$. This may cause some philosophical concern, but we'll leave it alone. If it does bother you, we can add a set of new constant symbols for all of the objects in the domain M and use them instead. The result is much the same, but more tiresome to wield.

With this in hand we can now describe how the satisfaction predicate works for sentences $\varphi \in Sent_{\mathcal{L}(M)}$ of the expanded language.

To make this work, we also need to *expand* the model $\mathcal{M}$ in order that it is a model of the expanded language $\mathcal{L}(M)$. To do this we simply let $m^{\mathcal{M}} = m$ for all $m \in M$. We shall denote the resultant model by $\mathcal{M}^+$.

Thus given a model $\mathcal{M}$, language $\mathcal{L}(M)$ and sentence $\varphi \in Sent_{\mathcal{L}(M)}$ we say that if φ is of the form $\exists x \psi(x)$, then

- $\mathcal{M}^+ \models \varphi$ iff there is some $m \in M$ such that $\mathcal{M}^+ \models \psi(m \mapsto x)$.

Since m is a constant symbol in $\mathcal{L}(M)$ and $\mathcal{M}^+$ is a model of $\mathcal{L}(M)$, it makes perfect sense to say $\mathcal{M}^+ \models \psi(m \mapsto x)$.

Universal quantification can be similarly characterised.

So putting it all together, we now have the means of telling whether a sentence $\varphi \in Sent_{\mathcal{L}(M)}$ is true in $\mathcal{M}^+$. But what we really want is a way of telling whether or not a sentence $\varphi \in Sent_{\mathcal{L}}$ is true in our original model $\mathcal{M}$.

We first observe that:

FACT 45. *if* $\varphi \in Sent_{\mathcal{L}}$, *then* $\varphi \in Sent_{\mathcal{L}(M)}$.

This tells us that sentence from $\mathcal{L}$ are also sentences in $\mathcal{L}(M)$, so it makes sense to say $\mathcal{M}^+ \models \varphi$ for $\varphi \in Sent_{\mathcal{L}}$. Moreover, we can just use $\mathcal{M}^+$.

DEFINITION 46. Given a language $\mathcal{L}$, $\mathcal{M}$ a model of $\mathcal{L}$ and $\varphi \in Sent_{\mathcal{L}}$

- $\mathcal{M} \models \varphi$ iff $\mathcal{M}^+ \models \varphi$.

This is what we wanted!

REMARK 47. Strictly, we also need another clause which says that nothing else is true in $\mathcal{M}$.

2.2.4.3. $=$ *and* $\bot$. $=$ is a two place relation which says that the object represented by the first constant symbol is identical to the object represented by the second constant symbol.

$\bot$ is a 0-place relation symbol which is false in every model.

See Exercise 53

2.3. Important Semantic Properties

In this final section, we are going to use models and our satisfaction relation to define some interesting properties that a sentence might have.

DEFINITION 48. φ is *satisfiable* if there is some model $\mathcal{M}$ in the language $\mathcal{L}$ of φ such that $\mathcal{M} \models \varphi$.

DEFINITION 49. φ is *valid* (or a *logical truth*), abbreviated $\models \varphi$, if for every model $\mathcal{M}$ in the language $\mathcal{L}$ of φ we have $\mathcal{M} \models \varphi$.

Intuitively speaking, φ is satisfiable if there is some way (i.e., a model) of making φ true. Similarly φ is valid if there is no way of making it false.

REMARK 50. Note that we use the symbol, $\models$, for both validity and satisfaction. This practice is called *overloading*. Context should always settle any confusion.

Let us write Γ, Δ for sets of sentences in some language $\mathcal{L}$; i.e., such that $\Gamma, \Delta \subseteq Sent_{\mathcal{L}}$.

DEFINITION 51. Given $\Gamma \subseteq Sent_{\mathcal{L}}$ and $\varphi \in Sent_{\mathcal{L}}$, we say that φ *is a consequence of* Γ, abbreviated $\Gamma \models \varphi$, if for every model $\mathcal{M}$ of $\mathcal{L}$ if we have $\mathcal{M} \models \gamma$ for ever $\gamma \in \Gamma$, then $\mathcal{M} \models \varphi$.

Intuitively, if φ is a consequence of Γ we are saying that every way of making all of Γ true is a way which also makes φ true. Thus, we might say that Γ implies φ.

Exercises Week 2

EXERCISE 52. Provide the clause defining the behaviour of sentences of the form $\forall x\psi(x) \in Sent_{\mathcal{L}(M)}$ for some model $\mathcal{M}$ and language $\mathcal{L}$.

EXERCISE 53. Provide the $\models$ clauses for $=$ and $\bot$.

EXERCISE 54. Write out the full definition of $\models$.

EXERCISE 55. Let $\mathcal{L} =\{a, P, R\}$ be the language where a is a constant symbol and P and R are relation symbols with arity 1 and 2 respectively. Let $\mathcal{M}$ be a model of $\mathcal{L}$ such that:

- $M = \{m_1, m_2\}$;
- $a^{\mathcal{M}} = m_2$;
- $P^{\mathcal{M}} = \{m_1\}$; and
- $R^{\mathcal{M}} = \{\langle m_1, m_1\rangle, \langle m_2, m_2\rangle\}$.

Using the satisfaction definition ($\models$) work out whether or not the following are correct:

(1) $\mathcal{M} \models Pa$;
(2) $\mathcal{M} \models Raa \vee \neg Pa$;
(3) $\mathcal{M} \not\models \neg\neg Ra$;
(4) $\mathcal{M} \models Pm_2$;
(5) $\mathcal{M}^+ \models Pm_2m_1$;
(6) $\mathcal{M} \models \forall x(Px \vee \neg Px)$;
(7) $\mathcal{M} \models \forall xRxx$; and
(8) $\mathcal{M} \models \forall x\exists y(Rxy \wedge Py)$.

EXERCISE 56. Consider the language $\mathcal{L}$ from the exercise above and the domain $M = \{m_1, m_2\}$. We can construct a different model of $\mathcal{L}$ be interpreting the nonlogical vocabulary differently. How may different models of $\mathcal{L}$ can you make on the domain M.

Given a model $\mathcal{M}$ whose domain has n many objects in it, which we shall write $|\mathcal{M}| = n$ and say that the cardinality of $\mathcal{M}$ is n. Suppose the language consists of one m-place relation symbol R, how many different interpretations (and thus models) are there for such a relation over $\mathcal{M}$?

EXERCISE 57. In Remark 47, we note that a closing off clause is required. What could happen if we didn't add one of these.

EXERCISE 58. Consider the standard model of arithmetic $\mathbb{N}$ in the language $\mathcal{L} = \{0, Prime, 1, +, \times\}$ from Example 34 with an expansion consisting of a new one-place relation symbol, $Prime$, which is true of a number iff it is prime. Allowing the "internal" notation for arithmetic functions, verify, if you can, whether the following are correct:

(1) $\mathbb{N} \models \neg(1 + 0 = 1)$;
(2) $\mathbb{N} \models \exists x(x = 1 + 1)$;
(3) $\mathbb{N} \models 1 + 1 = 2$;
(4) $\mathbb{N} \models \forall x(x = 0 \rightarrow \neg x = 1)$;
(5) $\mathbb{N} \models \forall x \forall y(x \times y = y \times x)$;
(6) $\mathbb{N} \models \forall x(Prime(x) \rightarrow \exists y(y \times (1 + 1) = x))$; and
(7) $\mathbb{N} \models \forall x(\exists y(y \times (1 + 1) = x) \rightarrow \exists y \exists z(Prime(y) \wedge Prime(z) \wedge x = z + y))$.

If you cannot verify it, explain why not.

EXERCISE 59. Let $M = \{m_1, m_2\}$ be a domain consisting of two objects m_1 and m_2. Consider the language $\mathcal{L} = \{R\}$ consisting of a single two place relation symbol. Now consider the following sentences:

(1) $\exists x Rxx$;
(2) $\exists x Rxx \wedge \exists y \neg Ryy$; and
(3) $\forall x Rxx$.

For each of these sentences write out the set of models over M such that that sentence is true there; i.e., for each sentence (1.)-(3.), define all of the interpretations of R such that that sentence would be true. What differences do you note? Consider the following questions:

- Is there more than one model?
- If there is more than one model, could we add another sentence such that we can pin down only one model?

EXERCISE 60. Verify Fact 45. [Hint: prove this by induction on the stage construction of formulae before considering sentences.]

EXERCISE 61. Define *satisfiable* in terms of *validity*; i.e., give a simple definition of what it means for a formula to be satisfiable using the concept of validity.

EXERCISE 62. Consider the following properties a sentence $\varphi \in Sent_{\mathcal{L}}$and set of sentence $\Gamma \subseteq Sent_{\mathcal{L}}$ might have:

(1) There is no model $\mathcal{M}$ such that $\mathcal{M} \models \gamma$ for all $\gamma \in \Gamma$ and $\mathcal{M} \not\models \varphi$.
(2) If for all models $\mathcal{M}$, $\mathcal{M} \models \gamma$ for all $\gamma \in \Gamma$, then for all models $\mathcal{M}$, $\mathcal{M} \models \varphi$.

Which of these, if any, is equivalent to saying $\Gamma \models \varphi$.

EXERCISE 63. Let $\mathcal{M}$ be a model of some language $\mathcal{L}$. Show that for any sentence φ from $\mathcal{L}$ that:

$$\mathcal{M} \models \varphi \;\Leftrightarrow\; \mathcal{M} \not\models \neg\varphi.$$

This, in effect, says that every sentence is either true or false in $\mathcal{M}$ but not both. [Hint: Prove this by induction on the complexity of formulae in the expanded language.]

CHAPTER 3

Proof Systems

Goals:

(1) To review tableau proof systems.
(2) Review natural deduction proof system.

This is mainly a review week. We'll move over this material quickly. It is expected that you already know how to use these systems.

Our goal this week is to explore two proof systems. In each system we want to be able to show whether:

- some $\varphi \in Sent_{\mathcal{L}}$ is a *theorem*; or
- φ may be derived from some (finite) $\Gamma \subseteq Sent_{\mathcal{L}}$.

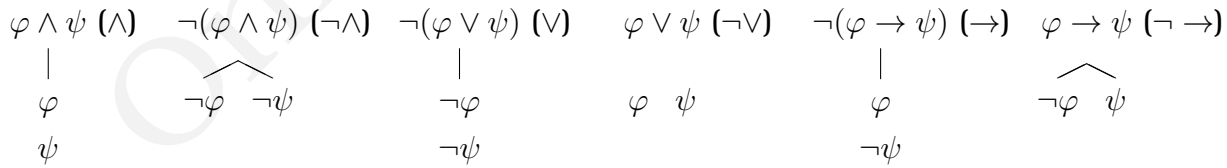
It turns out that the most convenient way of conduction these proofs requires an infinite supply of constant symbols C (sometimes known as parameters). As in the previous week, we shall denote the expanded language by $\mathcal{L}(C)$.

3.1. Tableau

3.1.1. Starting conditions. To establish whether φ is a *theorem*, we commence a tableau by placing $\neg\varphi$ at the top node of the tableau.

To establish whether φ may be *derived* from some (finite) $\Gamma \subseteq Sent_{\mathcal{L}}$, we commence the tableau by placing each of the sentence $\gamma \in \Gamma$ followed by $\neg\varphi$ at the top of the tableau.

3.1.2. Rules. The following tableau rules should be familiar.



$$\begin{array}{c}\neg\neg\varphi\ (\neg\neg)\\ |\\ \varphi\end{array}$$

$$\begin{array}{c}\neg\forall x\varphi(x)\ (\forall)\\ |\\ \neg\varphi(a)\end{array}\qquad\begin{array}{c}\forall x\varphi(x)\ (\neg\forall)\\ |\\ \varphi(t)\end{array}\qquad\begin{array}{c}\exists x\varphi(x)\ (\exists)\\ |\\ \varphi(a)\end{array}\qquad\begin{array}{c}\neg\exists x\varphi(x)\ (\neg\exists)\\ |\\ \neg\varphi(t)\end{array}$$

3.1.3. Closure conditions. A branch $\mathcal{B}$ is a tableau is deemed to be *closed* if for some sentence $\varphi \in \mathcal{L}(C)$ both φ and $\neg\varphi$ occur on $\mathcal{B}$. If every branch of a tableau closes, then the tableau is *closed*.

If the tableau for $\neg\varphi$ closes, then we have a *proof* of φ, abbreviated $\vdash_{Tab} \varphi$. We shall omit the $_{Tab}$ where no confusion can arise.

If the tableau commencing with $\Gamma, \neg\varphi$ closes, then we have derived φ from Γ, which we abbreviate $\Gamma \vdash_{Tab} \varphi$.

3.1.4. Examples.

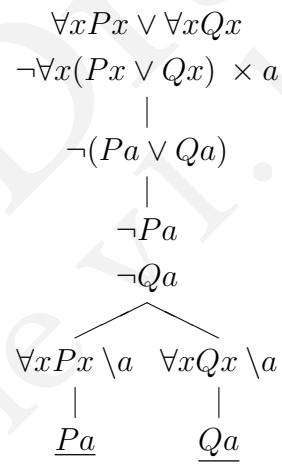
EXAMPLE 64. $\vdash \forall xPx \to \neg\exists x\neg Px$

$$\begin{array}{c}\neg(\forall xPx \to \neg\exists x\neg Px)\\ |\\ \forall xPx\ \backslash a\\ \neg\neg\exists x\neg Px\\ |\\ \exists x\neg Px\ \times a\\ |\\ \neg Pa\\ |\\ \underline{Pa}\end{array}$$

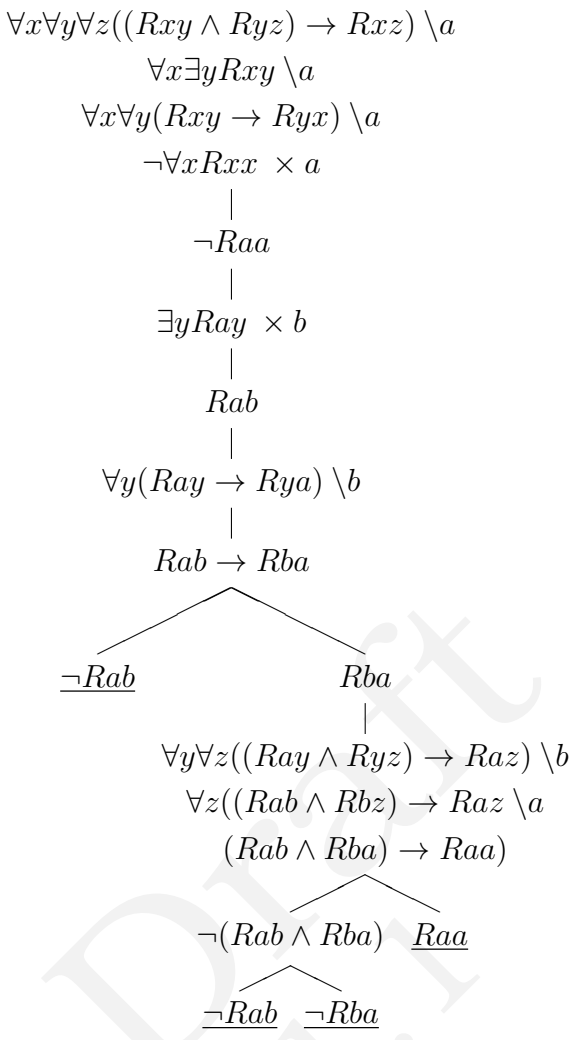
EXAMPLE 65. $\vdash \neg\exists x\neg Px \to \forall xPx$

$$\neg(\neg\exists x\neg Px \to \forall xPx)$$
$$|$$
$$\neg\exists x\neg Px \setminus a$$
$$\neg\forall xPx \ \times a$$
$$|$$
$$\neg Pa$$
$$|$$
$$\underline{\neg\neg Pa}$$

EXAMPLE 66. $\forall xPx \vee \forall xQx \vdash \forall x(Px \vee Qx)$

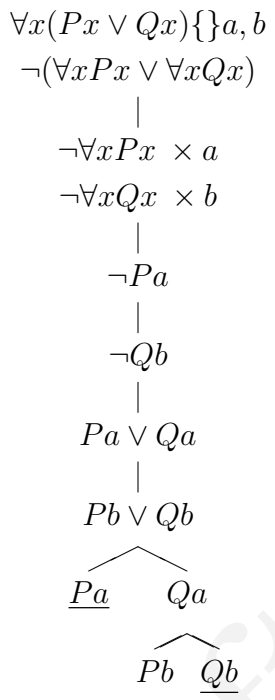


EXAMPLE 67. $\forall x\forall y\forall z((Rxy \wedge Ryz) \to Rxz), \forall x\exists yRxy, \forall x\forall y(Rxy \to Ryz) \vdash \forall xRxx$



3.1.5. Counterexamples.

EXAMPLE 68. $\forall x(Px \vee Qx) \nvdash \forall x Px \vee \forall x Qx$



We have applied all the rules we needed to, but there is still an open branch. Thus there is no derivation of $\forall xPx \vee \forall xQx$ from $\forall x(Px \vee Qx)$.

We now use the open branch to construct a model which *witnesses* a counterexample to the derivation; i.e., a model in which all of the premises are true but the conclusion is false.

We let $\mathcal{M} = \langle M, P^{\mathcal{M}}, Q^{\mathcal{M}} \rangle$ be as follows:

- M is the set of terms occurring on the open branch (i.e., $\{a, b\}$);
- $P^{\mathcal{M}}$ is the set of terms t that occur in sentences of the form Pt that occur on the open branch (i.e., $\{b\}$);
- $Q^{\mathcal{M}}$is the set of terms t that occur in sentences of the form Qt that occur on the open branch (i.e., $\{a\}$).

Now it should be clear that we have $\mathcal{M} \models \forall x(Px \vee Qx)$ but $\mathcal{M} \not\models \forall xPx \vee \forall xQx$.

EXAMPLE 69. $\nvdash \forall x \exists y Rxy$

$$\begin{array}{c} \neg\exists x\forall y Rxy\{\}a,b \\ | \\ \neg\forall y Ray \times b \\ | \\ \neg Rab \\ | \\ \neg\forall y Rby \times c \\ | \\ Rbc \end{array}$$

Now it should be clear that this tableau is going to go on forever. This clearly means that the branch (there's only one) will remain open. But how do we describe a model which makes $\forall x\exists y Rxy$ false.

In this case, we can think up a model which will do the trick without using the branch. Let $\mathcal{M} = \langle M, R^{\mathcal{M}}\rangle$ be such that:

- $M = \{a\}$; and
- $R^{\mathcal{M}} = \{\}$ (i.e., nothing is related by R.

Now you should be able to see that $\mathcal{M} \models \forall x\exists y Rxy$.

3.2. Natural Deduction

3.2.1. Rules. Again, these rules should be familiar from the course on philosophical logic.

$$\frac{\varphi \qquad \psi}{\varphi \wedge \psi}\ (\wedge\text{-I}) \qquad \frac{\varphi \wedge \psi}{\varphi}\ (\wedge\text{-E}) \qquad \frac{\varphi \wedge \psi}{\psi}\ (\wedge\text{-E})$$

$$\frac{\varphi}{\varphi \vee \psi}\ (\vee\text{-I}) \qquad \frac{\psi}{\varphi \vee \psi}\ (\vee\text{-I}) \qquad \frac{\varphi \vee \psi \qquad \begin{array}{c}(\varphi)\\ \chi\end{array} \qquad \begin{array}{c}(\psi)\\ \chi\end{array}}{\chi}\ (\vee\text{-E})$$

$$\frac{\begin{array}{c}(\varphi)\\ \psi\end{array}}{\varphi \to \psi}\ (\to\text{-I}) \qquad \frac{\varphi \qquad \varphi \to \psi}{\psi}\ (\to\text{-E})$$

$$\frac{\begin{array}{c}(\varphi)\\ \bot\end{array}}{\neg\varphi}\ (\neg\text{-I}) \qquad \frac{\neg\varphi \qquad \varphi}{\bot}\ (\neg\text{-E}) \qquad \frac{\neg\neg\varphi}{\varphi}\ (\text{DN})$$

$$\frac{\varphi(a)}{\forall x\varphi(x)}\,(\forall\text{-I}) \qquad \frac{\forall x\varphi(x)}{\varphi(t)}\,(\forall\text{-E})$$

$$\frac{\varphi(t)}{\exists x\varphi(x)}\,(\exists\text{-I}) \qquad \frac{\exists x\varphi(x) \qquad \begin{matrix}\varphi(a)\\ \psi\end{matrix}}{\psi}\,(\exists\text{-E})$$

(∀-I) Provided a is not free in $\forall x\varphi(x)$ nor in any assumption occurring at or above $\varphi(a)$. (∃-E) Provided that a is not free in any assumption in the right branch nor in $\exists x\varphi(x)$ nor ψ.

3.2.2. Derivations and theorems.

DEFINITION 70. We say that φ can be *derived from assumptions* in Γ, abbreviated $\Gamma \vdash_{Nat} \varphi$, when there is natural deduction proof with φ on the final line and whose assumptions are all members of Γ. We say that φ is a *theorem* if $\vdash_{Nat} \varphi$: i.e., φ can be derived without any assumptions.

I'm going to use a slightly slicker approach to writing up natural deduction proofs. I'll give you a simple example which I'll write up in the style you've already learned.

3.2.3. Examples.

EXAMPLE 71. $Pa \to Qa \vdash \neg Qa \to \neg Pa$

$$\dfrac{\dfrac{\dfrac{Pa^{(1)} \qquad Pa \to Qa}{Qa} \qquad \neg Qa^{(2)}}{\dfrac{\bot}{\neg Pa}\,(1)}}{\neg Qa \to \neg Pa}\,(2)$$

We can write out the same derivation in the old notation as follows:

1	(1) $Pa \to Qa$	Premise
2	(2)Pa	Assumption
1,2	(3) Qa	→-E (1, 2)
4	(4) $\neg Qa$	Assumption
1,2,4	(5) $\bot$	⊥-I (3,4)
1,4	(6) $\neg Pa$	¬-I (5,2)
1	(7) $\neg Qa \to \neg Pa$	→-I (6,4)

The only difference is that the book-keeping is a little lighter and hopefully, you'll see that it's a lot easier to see what's going on.

3.3. Interesting Properties

DEFINITION 72. (Natural deduction) A set Γ of sentences is *consistent* if there is no way to derive $\bot$ from Γ; i.e., $\Gamma \nvdash \bot$. (Tableau) A set Γ is *consistent* if for every tableau commencing with a (finite) subset of Γ remains open.

THEOREM 73. *(The deduction theorem)* $\Gamma \cup \{\varphi\} \vdash \psi$ *iff* $\Gamma \vdash \varphi \to \psi$.

PROOF. For the tableau systems both proofs start with almost exactly the same set up. We leave this as an exercise.

Let's consider the natural deduction system. Suppose that we have a derivation of $\Gamma \cup \{\varphi\} \vdash \psi$. Then clearly we may add another line to the derivation, which employs $\to$-I and thus gives us $\Gamma \vdash \varphi \to \psi$.

On the other hand, suppose we have a derivation of $\Gamma \vdash \varphi \to \psi$. Then suppose we add a new line to that derivation, which employs $\to$-E and the added assumption φ. Then we clearly end up with a derivation of ψ from $\Gamma \cup \{\varphi\}$. □

REMARK 74. In other proof systems, the deduction theorem can be quite painful to prove.

THEOREM 75. *(Cut/Lemma - Natural deduction only) Suppose* $\Gamma \vdash \varphi$ *and* $\Gamma \cup \{\varphi\} \vdash \psi$. *Then* $\Gamma \vdash \psi$.

PROOF. Suppose we have $\Gamma \vdash \varphi$ and $\Gamma \cup \{\varphi\} \vdash \psi$. Then by the deduction theorem (Theorem 73) we have $\Gamma \vdash \varphi \to \psi$. Thus we may combine the derivations of φ and $\varphi \to \psi$ from Γ to get a derivation of ψ from Γ using $\to$-E. □

THEOREM 76. *(Monotonicity) Suppose* $\Gamma \vdash \varphi$ *and* $\Delta \supseteq \Gamma$, *then* $\Delta \vdash \varphi$.

Exercises Week 3

EXERCISE 77. Describe tableau rules and natural deduction rules for identity $=$.

EXERCISE 78. Complete the following derivations, if possible, in both the tableau system and the natural deduction system. You may want to use the cut lemma or some derive some new rules. If it is not correct provide a counterexample:

(1) $Pa \to (Qa \to Rc) \dashv\vdash (Pa \wedge Qa) \to Rc$;
(2) $Pa \to Qb \dashv\vdash \neg Qb \to \neg Pa$;
(3) $Qb \vdash Pa \to Qb$;
(4) $\neg Pa \vdash Pa \to Qb$;
(5) $\neg(Pa \to Qb) \dashv\vdash Pa \wedge \neg Qb$;
(6) $Pa \to Qb \dashv\vdash \neg Pa \vee Qb$;
(7) $\vdash Pa \vee \neg Pa$;
(8) $\neg Pa \wedge \neg Qb \dashv\vdash \neg(Pa \vee Qb)$;
(9) $\neg Pa \vee \neg Qb \dashv\vdash \neg(Pa \wedge Qb)$;
(10) $\forall x(Px \wedge Qx) \dashv\vdash \forall x Qx \wedge \forall x Px$;
(11) $\forall x(Px \to Qx), \forall x(Qx \to Rx), Pa \vdash Ra$;
(12) $Pa \wedge (Qb \vee Rc) \dashv\vdash (Pa \wedge Qb) \vee (Pa \wedge Rc)$;
(13) $Pa \vee (Qb \wedge Rc) \dashv\vdash (Pa \vee Qb) \wedge (Pa \vee Rc)$;
(14) $\forall x(Px \to Qa) \dashv\vdash \exists x Px \to Qa$;
(15) $\exists x(Px \to Qa) \dashv\vdash \forall x Px \to Qa$;
(16) $\forall x(Qa \to Px) \dashv\vdash Qa \to \forall x Px$;
(17) $\exists x(Qa \to Px) \dashv\vdash Qa \to \exists x Px$;
(18) $\forall x \forall y(Rxy \to \neg Rxy) \vdash \forall x \neg Rxx$;
(19) $\forall x \forall z(\exists y(Rxy \wedge Ryz) \to Rxz), \forall x \forall y(Rxy \to Ryx) \vdash \forall x \forall y(Rxy \to Rxx)$;
(20) $\forall x \forall z \forall y((Rxy \wedge Ryz) \to Rxz), \forall x \forall y(Rxy \to Ryx) \vdash \forall x Rxx$.

($\dashv\vdash$ means demonstrate both directions.)

EXERCISE 79. If we could derive $\varphi \wedge \neg\varphi$ from Γ, is Γ consistent? Show that $\varphi \wedge \neg\varphi \vdash \psi$ for any sentences φ, ψ. Show that $\bot \vdash \psi$ for all sentences ψ. Why would this be a bad situation? [Perhaps Γ is a theory which we hold to be true. How useful would it be?]

EXERCISE 80. Prove Theorem 73 and Theorem 76 for the tableau case.

EXERCISE 81. Consider the set Γ of sentences consisting of:

- $\forall x \forall y \forall z(Rxy \wedge Ryz \rightarrow Rxz)$
- $\forall x \forall y(Rxy \rightarrow \neg Ryx)$
- $\forall x \exists y Rxy$

Is this set of sentences consistent? Describe a model for it. Can you find a finite model? Why/Why not?

EXERCISE 82. Can you prove Theorem 75 for the tableau case? What goes wrong?

CHAPTER 4

Completeness 1

Goals:

- Prove the soundness of the tableau proof system.
- Prove the completeness of the tableau proof system.

Our goal is to demonstrate the following theorem.

THEOREM 83. $\vdash_{Tab} \varphi$ *iff* $\models \varphi$. [1]

The ($\rightarrow$) direction is known as *soundness*; and the ($\leftarrow$) direction is known as completeness.

To keep things simple, we shall not worry (this week) about consequences and derivations. The theorem is thus often known as *weak completeness*, since we have not involved a set Γ of sentences.

We shall also avoid the use of function symbols.

4.1. Completeness

4.1.1. The Strategy. Our goal is to show that if $\models \varphi$, then $\vdash \varphi$. Or in other words, if every model $\mathcal{M}$ (of the language of φ) is such that $\mathcal{M} \models \varphi$, then the tableau commencing with $\neg\varphi$ is closed.

It's hard to know how to start here though. How could we take the fact that very model makes φ true, to showing that there is a proof of φ. Perhaps we can *re-articulate* the question in a way that is more tractable.

For example, we already know that:

FACT 84. *(Contraposition)* $\varphi \rightarrow \psi$ *iff* $\neg\psi \rightarrow \neg\varphi$.

Thus, we may voice the completeness problem as follows:

- If $\nvdash \varphi$, then $\not\models \varphi$

or in other words,

[1] For convenience, I'll stop using the $_{Tab}$ for the remainder of this week's notes.

- if the tableau commencing with $\neg\varphi$ does not close (i.e., it has at least one open branch), then there is some $\mathcal{M}$ such that $\mathcal{M} \models \neg\varphi$.

This kind of thing should be more familiar. We have seen examples of how to take an open branch of a tableau and define a model from it: this is how we constructed counterexamples.

Of course, this only worked in *finite* cases. Our goal in this section will be to *generalise* this technique to the case of *infinite branches*.

4.1.1.1. *Overview.* At a high level, we are taking the fact that there is no proof of φ and using that fact to make a set of sentences which describes a model in which $\neg\varphi$ is true.

Our strategy can be described as follows:

(1) Find a *special set of sentences* $\mathcal{B}$ to describe the model (this is the open branch);
(2) Define a *model* $\mathcal{M}^{\mathcal{B}}$ using the special set; and
(3) Show that $\neg\varphi$ is indeed true in $\mathcal{M}^{\mathcal{B}}$.

This will clearly suffice to establish our completeness theorem.

We shall use the same strategy again next week when the proof will be more difficult. However, this provides a good template for seeing what is going on.

4.1.2. The Proof. We shall proceed, first, in the language without the identity relation $=$.

4.1.2.1. *The special set.* We start with the fact that $\nvdash \varphi$. This means that the tableau commencing with $\neg\varphi$ does not close. Thus there is at least one open branch in the tableau. Pick one and call it $\mathcal{B}$.

Our special set is the collection of sentences occurring on $\mathcal{B}$.

4.1.2.2. *Defining the model.* To define a model, we are going to need need:

- a language;
- a domain; and
- an interpretation.

The language $\mathcal{L}$ of the model will be $\mathcal{L}(C)$, which is the language from which φ came (i.e., $\mathcal{L}$) augmented with the constant symbols c required for the construction of the tableau.

We then define our model $\mathcal{M}^{\mathcal{B}}$ as follows:

- Let the domain $M^{\mathcal{B}}$ be the set of constant symbols occurring in sentences on $\mathcal{B}$.
- For each n-ary relation symbol R from $\mathcal{L}(C)$, let $R^{\mathcal{M}^{\mathcal{B}}}$ (the interpretation of R in $\mathcal{M}^{\mathcal{B}}$) be the set of n-tuples $\langle a_1, ..., a_n \rangle$ of constant symbols such that the sentence $Ra_1, ..., a_n$ occurs on $\mathcal{B}$.
- For each constant symbol a from $\mathcal{L}(C)$, let $a^{\mathcal{M}^{\mathcal{B}}}$ be a; i.e., *itself*.

These are all the ingredients require for a model, so the job of defining one is complete. The remaining job is to demonstrate that it does what we want.

REMARK 85. Note that we have used the constant symbols themselves to form the domain. The symbols are objects too, so there is nothing wrong with doing this.

4.1.2.3. *Showing that* $\mathcal{M}^{\mathcal{B}} \models \neg\varphi$. Up to here, everything has been quite straightforward. Establishing this fact, is a *little* more tricky.

First, we note that, intuitively speaking, we ought to think that this is right. We've been constructing finite counterexamples models using this technique, and this is just a simple generalisation of that approach. There are a couple of things to say here. First, just because it worked in the finite case, doesn't necessarily mean it will work with infinite branches. We need to establish this. Second, when we used the counterexample construction technique for tableau, we noted that it worked in specific cases, but we didn't really prove that it worked. That is what we're going to do now.

So our goal is to show that $\neg\varphi$ is true in $\mathcal{M}^{\mathcal{B}}$. It's hard to know how we'd get straight to such a fact. However, our construction of $\mathcal{M}^{\mathcal{B}}$ leads to a fact and a reasonable hypothesis:

- **FACT**: If an atomic sentence or its negation is on $\mathcal{B}$, then that sentence will be true in $\mathcal{M}^{\mathcal{B}}$. We shall check this again later, but you should see that this is just what we've ensured by our definition of the model $\mathcal{M}^{\mathcal{B}}$.
- **HYPOTHESIS**: Perhaps all of the sentences on $\mathcal{B}$ are true in $\mathcal{M}^{\mathcal{B}}$.

We might suppose that the hypothesis is correct given the way that the rules for the construction of a tableau work. Thus we might propose the following:

LEMMA 86. *If* ψ *is on* $\mathcal{B}$*, then* $\mathcal{M}^{\mathcal{B}} \models \psi$.

Clearly, if Lemma 86 is correct, we would have established that $\mathcal{M}^{\mathcal{B}} \models \neg\varphi$, since $\neg\varphi$ is on $\mathcal{B}$.

This is what we shall demonstrate.

Now this is a universal claim: we are showing that

- every ψ is a member of the set of sentences A.

where A is the set of ψ such that if ψ is on $\mathcal{B}$, then $\mathcal{M}^{\mathcal{B}} \models \psi$.

From Week 1, we know a technique that might be useful for establishing such a claim: induction.

Moreover, the way in which the tableau is constructed might *hint* at a kind of inductive *flavour*.

So we might return to our recursive construction of the well-formed formulae WFF. Perhaps we can show that if all the sentences constructed by stage n in the construction of WFF are in A, then all the sentences constructed at stage $n+1$ are also in A. From here we could then employ the principle of induction and establish that every $\psi \in WFF$ is also in A. This would establish the lemma.

However, there is a problem. The usual way of building up the well-formed formulae in stages (as we did in Week 1) does not work. In the remainder of this subsection, I will:

(1) show our usual construction does not work;
(2) develop a new stage-based construction of the well-formed formulae; and
(3) complete the proof of the lemma on the basis of the alternative construction.

You may prefer to skip (1.) on a first reading.

So suppose we try to use the stage construction from Week 1. Here then is how the proof might go:

PROOF. (**ABORTIVE**) We proceed by induction on the stages of construction of well-formed formulae.

(Base) If ψ is atomic and on $\mathcal{B}$, then ψ is of the form $Ra_1, ..., a_n$ for some n-ary relation symbol R and constant symbols $a_1, ..., a_n$. Then we have

$$\begin{aligned} Ra_1...a_n \text{ is on } \mathcal{B} &\Leftrightarrow \langle a_1^{\mathcal{M}^{\mathcal{B}}}, ..., a_n^{\mathcal{M}^{\mathcal{B}}} \rangle \in R^{\mathcal{M}^{\mathcal{B}}} \\ &\Leftrightarrow \mathcal{M}^{\mathcal{B}} \models Ra_1...a_n \end{aligned}$$

The first $\Leftrightarrow$ follows from the definition of $\mathcal{M}^{\mathcal{B}}$; and the second, from the definition of $\models$.

REMARK. So clearly the atomic case works. Let's try the induction step.

(Induction Step) Suppose that for all sentences χ of complexity $\leq n$ we have established that:

- if χ is on $\mathcal{B}$, then $\mathcal{M}^{\mathcal{B}} \models \chi$.

This is our *induction hypothesis*.

We show that the same is true for sentences ψ of complexity $n+1$. Tho do this we consider all of the forms of formulae that could be constructed from formulae of complexity $\leq n$.

Suppose $\psi := \chi \wedge \delta$ is on $\mathcal{B}$ where χ and δ have complexity $\leq n$. Then by the rules for construction of tableau, both χ and δ occur on $\mathcal{B}$. By our induction hypothesis, we see that $\mathcal{M}^{\mathcal{B}} \models \chi$ and $\mathcal{M}^{\mathcal{B}} \models \delta$; and thus by the definition of $\models$, we get $\mathcal{M}^{\mathcal{B}} \models \chi \wedge \delta$.

REMARK. Thus the induction step works for conjunction. Perhaps unsurprisingly, the problem crops up with negation.

Suppose $\psi := \neg\chi$ is on $\mathcal{B}$ where χ has complexity $\leq n$. Since $\mathcal{B}$ is an open branch we see that χ is not on $\mathcal{B}$ ... □

But then what? We cannot make use of the induction hypothesis: it only gives us facts about $\mathcal{M}^{\mathcal{B}}$ given sentences *are on* $\mathcal{M}^{\mathcal{B}}$; it doesn't say anything about what to do when some χ is not on $\mathcal{B}$. So we are *stuck.*

Fortunately there is a way around things. To get the idea, we might look to how the case for conjunction worked:

(1) We used the *tableau rules* to observe that less complex sentences where already *on* $\mathcal{B}$; and
(2) From there we employed the induction hypothesis and the satisfaction definition.

We were held up because we couldn't get to another fact about what was *already on $\mathcal{B}$*.

We then observe that:

- use of the tableau always takes from facts about what is on $\mathcal{B}$ to less complex things on $\mathcal{B}$; and
- every sentence ψ has the form of the top sentences in one of the tableau rules.

The first of these suggests that if we only considered sentences of the form given in tableau rules, our induction would not get stuck. The second suggests that we may be able to define all of the well-formed formulae using just the forms given in tableau rules. It turns out that this is correct.

We now provide an alternative stage-by-stage construction of WFF and then show that the kind of complexity it gives, will get us through the proof of Lemma 86.

Let us define our stages as follows:

- $Stage_{WFF+}(0)$ is the set of formulae of the form $Rt_1, ...t_n$ or $\neg Rt_1, ..., t_n$ where R is an n-ary relation symbol and $t_1, ..., t_n$ are terms of the language (these formulae are sometimes know as *literals*);
- $Stage_{WFF+}(n+1)$ is the set of formulae
  - ψ such that $\psi \in Stage_{WFF+}(n)$;
  - of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\neg\varphi$, $\forall x\varphi(x)$ or $\exists x\psi(x)$ where $\varphi, \psi \in Sent_{WFF+}(n)$;
  - $\neg(\varphi \wedge \chi)$, $\neg(\varphi \vee \psi)$, $\neg\forall\varphi(x)$ or $\neg\exists x\varphi(x)$ where $\neg\varphi$ and $\neg\psi$ are in $Sent_{WFF+}(n)$; and
  - $\neg(\varphi \to \psi)$, $\psi \to \varphi$ where φ and $\neg\psi$ are in $Sent_{WFF+}(n)$.

We shall then say that the $+$-complexity of some φ is the least n such that $\varphi \in Stage_{WFF+}(n)$.

REMARK 87. Observe that in the first condition in the induction step, each formula is of the form of the top formulae in a tableau rule. This is where I took them from. You should figure out which rules give rise to to which clauses in the definition above.

It should then be relatively easy to see that:

FACT 88. *φ is a well formed formula ($\varphi \in WFF$) iff there is some n such that $\varphi \in Stage_{WFF^+}(n)$.*

REMARK. This itself would be a simple proof by induction.

With this in hand, we are finally ready to complete the proof of Lemma 86, which we restate for convenience.

LEMMA. **86** *If ψ is on $\mathcal{B}$, then $\mathcal{M}^{\mathcal{B}} \models \psi$.*

PROOF. We proceed by induction on the $+$-complexity of formulae.

(Base) If ψ is atomic and on $\mathcal{B}$, then ψ is of the form $Ra_1, ..., a_n$ for some n-ary relation symbol R and constant symbols $a_1, ..., a_n$. Then we have

$$\begin{aligned} Ra_1...a_n \text{ is on } \mathcal{B} &\Leftrightarrow \langle a_1^{\mathcal{M}^{\mathcal{B}}}, ..., a_n^{\mathcal{M}^{\mathcal{B}}} \rangle \in R^{\mathcal{M}^{\mathcal{B}}} \\ &\Leftrightarrow \mathcal{M}^{\mathcal{B}} \models Ra_1...a_n \end{aligned}$$

The first $\Leftrightarrow$ follows from the definition of $\mathcal{M}^{\mathcal{B}}$; and the second, from the definition of $\models$.

(Induction Step) Suppose that for all sentences χ of $+$-complexity $\leq n$ we have established that:

- if χ is on $\mathcal{B}$, then $\mathcal{M}^{\mathcal{B}} \models \chi$.

This is our *induction hypothesis*.

We show that the same is true for sentences ψ of $+$-complexity $n+1$. To do this we consider all of the forms of formulae that could be constructed from formulae of $+$-complexity $\leq n$.

Suppose $\psi := \chi \wedge \delta$ is on $\mathcal{B}$ where χ and δ have $+$-complexity $\leq n$. Then by the rules for construction of tableau, both χ and δ occur on $\mathcal{B}$. By our induction hypothesis, we see that $\mathcal{M}^{\mathcal{B}} \models \chi$ and $\mathcal{M}^{\mathcal{B}} \models \delta$; and thus by the definition of $\models$, we get $\mathcal{M}^{\mathcal{B}} \models \chi \wedge \delta$.

Suppose $\neg(\chi \wedge \delta)$ is on $\mathcal{B}$ where $\neg\chi$ and $\neg\delta$ have $+$-complexity $\leq n$. Then by the tableau construction rules, at least one of $\neg\chi$ and $\neg\delta$ is on $\mathcal{B}$. Suppose $\neg\chi$ is on $\mathcal{B}$. Then by induction hypothesis, $\mathcal{M}^{\mathcal{B}} \models \neg\chi$; and thus $\mathcal{M}^{\mathcal{B}} \models (\neg\chi \wedge \delta)$. Suppose $\neg\delta$ is on $\mathcal{B}$. Then by induction hypothesis, $\mathcal{M}^{\mathcal{B}} \models \neg\delta$; and thus $\mathcal{M}^{\mathcal{B}} \models \neg(\chi \wedge \delta)$.

The cases for $\neg(\chi \vee \delta)$, $\chi \vee \delta$, $\neg(\chi \to \delta)$, $\chi \to \delta$ are similar.

Suppose $\neg\neg\chi$ is on $\mathcal{B}$ where χ has $+$-complexity $\leq n$. Then by the rules of construction of tableau, χ is also on $\mathcal{B}$. Then by the induction hypothesis, we have $\mathcal{M}^{\mathcal{B}} \models \chi$; and thus, by the $\models$ definition, $\mathcal{M}^{\mathcal{B}} \models \neg\neg\chi$.

Suppose $\forall x\chi(x)$ is on $\mathcal{B}$ where $\chi(x)$ has $+$-complexity $\leq n$. Then by the tableau rules, for every constant a in $\mathcal{L}(C)$ we have $\chi(a)$ on $\mathcal{B}$. By induction hypothesis, we have $\mathcal{M}^{\mathcal{B}} \models \chi(a)$ for each $a \in \mathcal{L}(C)$. Then since the domain, $M^{\mathcal{B}}$, is just the set of constant symbols in $\mathcal{L}(C)$, we have $\mathcal{M}^{\mathcal{B}} \models \forall x\chi(x)$.

Suppose $\neg\forall x\chi(x)$ is on $\mathcal{B}$ where $\neg\chi(x)$ has $+$-complexity $\leq n$. then by the tableau rules, there is some constant symbol a in $\mathcal{L}(C)$ such that $\neg\chi(a)$ is on $\mathcal{B}$. Fix such an a. Then by induction hypothesis, we have $\mathcal{M}^{\mathcal{B}} \models \chi(a)$; and thus $\mathcal{M}^{\mathcal{B}} \models \exists x\chi(x)$.

The cases for $\exists x\chi(x)$ or $\neg\exists x\chi(x)$ are similar. □

4.1.2.4. *Adding the identity (=) relation.*

4.2. Soundness

4.2.1. The Strategy. Our goal is to show that if $\vdash \varphi$, then $\models \varphi$. In other words, we want to show that if the tableau commencing with $\neg\varphi$ is closed, then φ is true in every model.

We have a similar problem to the situation with completeness here. Given a tableau proof of some φ, how should we go about showing that φ is true in every model.

Again we are going to contrapose. Thus we show that:

- If $\nvDash \varphi$, then $\nvdash \varphi$.

Or, unpacking things a little, if there is a model $\mathcal{M}$ where $\mathcal{M} \models \neg\varphi$, then there is a tableau commencing with $\neg\varphi$ which has an open branch. So all we need to do is figure out how to find such a branch.

The key to doing this will be the model $\mathcal{M}$. It will, so to speak, provide us with a map through the tree.

Our proof requires two parts:

(1) We define a *process* for finding of the branch; and
(2) We show that the branch cannot *close.*

4.2.2. The Proof.

4.2.2.1. *Constructing the branch $\mathcal{B}$.* Suppose we have some sentence φ and a model $\mathcal{M} \models \neg\varphi$. Let $\mathcal{T}$ be the completed tableau for $\neg\varphi$. Such a tableau is one in in which every branch is either:

- closed; or
- such that every rule that can be used has been used.

Essentially, we are going to show how to find an open branch $\mathcal{B}$ in $\mathcal{T}$ by working our way step by step down it.

We shall ensure that the branch remains open by making sure that every sentence we add by the rules is actually true in $\mathcal{M}$. Since we know there are no sentences φ such that both it and its negation are true in $\mathcal{M}$ we are thus guaranteed to get an open branch.

Let us let the *length* of a branch $\mathcal{B}$ in a tableau $\mathcal{T}$ be the number of applications of tableau rules that have occurred in its construction.

LEMMA 89. *(Branch Extension) Suppose $\mathcal{B}$ is an initial branch of some completed tableau $\mathcal{T}$ and $\mathcal{M}$ is such that for all ψ on $\mathcal{B}$*

$$\mathcal{M} \models \psi.$$

Then provided there are still rules left to apply, there is some extension $\mathcal{B}'$ of branch $\mathcal{B}$ in $\mathcal{T}$ and expansion $\mathcal{M}'$ of $\mathcal{M}$ such that for all ψ on $\mathcal{B}'$

$$\mathcal{M}' \models \psi.$$

PROOF. To establish this, we need to go through each of the rules and show that there is a way to extend $\mathcal{B}$ such that every sentence on $\mathcal{B}'$ remains true in $\mathcal{M}$.

($\wedge$) Suppose the $\mathcal{B}$ continues with an application of the ($\wedge$) rule. Then $\chi \wedge \delta$ occurs at some earlier point on the branch $\mathcal{B}$ and thus $\mathcal{M} \models \chi \wedge \delta$ (by hypothesis). Thus $\mathcal{B}$ must be continued to $\mathcal{B}'$ be adding both χ and δ to the bottom of $\mathcal{B}$. We let $\mathcal{M}' = \mathcal{M}$ be the trivial expansion of $\mathcal{M}$. Moreover, $\mathcal{M} \models \chi$ and $\mathcal{M} \models \delta$.

($\vee$) Suppose $\mathcal{B}$ continues with an application of the ($\vee$) rule. Then $\mathcal{B}$ splits into a χ path and ψ path; and $\chi \vee \delta$ occurs at some earlier stage on $\mathcal{B}$. By the hypothesis, we have $\mathcal{M} \models \chi \vee \delta$ and so either $\mathcal{M} \models \chi$ or $\mathcal{M} \models \delta$. Continue $\mathcal{B}$ to $\mathcal{B}'$ by picking a side of the fork on which the formula (either χ or δ) is true in $\mathcal{M}$ (it could be both). Then letting $\mathcal{M}' = \mathcal{M}$, we have our result. The cases for ($\vee$), ($\neg\vee$), ($\rightarrow$), ($\neg\rightarrow$) and ($\neg\neg$) are left as exercises.

($\exists$) Suppose $\mathcal{B}$ continues with an application of the ($\exists$) rule. Then, by the tableau rules, the next sentence on $\mathcal{B}_{n+1}$ is something of the form $\varphi(a \mapsto m)$ where a does not occur in any sentence already on $\mathcal{B}$; and $\exists x\varphi(x)$ occurs on $\mathcal{B}$ at some earlier stage. By the hypothesis, we have $\mathcal{M} \models \exists x\varphi(x)$, so there is some $m \in M$ such that $\mathcal{M}^+ \models \varphi(m \mapsto x)$. Let $\mathcal{M}'$ an interpretation of $\mathcal{L} \cup \{a\}$ such that $a^{\mathcal{M}} = m$. Then $\mathcal{M}' \models \varphi(a \mapsto n)$ as required.

($\forall$) Suppose $\mathcal{B}_n$ continues with an application of the ($\forall$) rule. Then $\forall x\varphi(x)$ occurs on $\mathcal{B}_n$ at some earlier stage and the next sentence on $\mathcal{B}_n$ is something of the form $\varphi(a)$ for some constant symbol

- occurring in some sentence higher up the branch $\mathcal{B}_n$; or
- an arbitrary constant symbol a from C if there are no constant symbols occurring above $\forall x\varphi(x)$.

In either case, by the hypothesis, we then have that $\mathcal{M} \models \forall x\varphi(x)$ and thus $\mathcal{M} \models \varphi(a \mapsto x)$ for any constant symbol of the language. In the former case, we let $\mathcal{M}' = \mathcal{M}$ and we are done. In the latter case, we let $\mathcal{M}'$ be a model of the language $\mathcal{L} \cup \{a\}$ such that $a^{\mathcal{M}} = m$ from some arbitrary $m \in M$. Then we have $\mathcal{M}' \models \varphi(a \mapsto x)$. The cases for ($\neg\forall$) and ($\neg\exists$) are left as exercises. □

LEMMA 90. *Suppose $\mathcal{M} \models \neg\varphi$. Then there is an open branch in the tableau $\mathcal{T}$ for φ.*

PROOF. We prove this in two stages:

(1) we describe how to construct the open branch;
(2) we demonstrate that it is open.

To construct the open branch we commence the tableau in the usual way by placing $\neg\varphi$ at the top of the tableau. To find the next stage of the branch we apply the Branch Extension Lemma (89). To find the stage after than we apply the Branch Extension Lemma again and so on. We stop if there are no more rules left to apply and if this never happens we keep going (thus leaving an infinite branch). Observe that this is just an informal version of a definition by recursion. We could do this more formally by setting up a system of stages.

Now we show that the branch $\mathcal{B}$ resulting from this process is open. Suppose not. Then $\mathcal{B}$ must be closed. Thus at some (finite) stage, say $\mathcal{B}_n$, in the construction of $\mathcal{B}$ we must have both ψ and $\neg\psi$ on $\mathcal{B}_n$. But our use of the

Branch Extension Lemma tells us that for every sentence χ on $\mathcal{B}_n$ we have $\mathcal{M}_n \models \chi$. Thus we have $\mathcal{M} \models \psi$ and $\mathcal{M} \not\models \psi$ which is a contradiction. □

THEOREM 91. *If* $\vdash \varphi$, *then* $\models \varphi$.

PROOF. Suppose $\not\models \varphi$. Then there is some model $\mathcal{M}$ such that $\mathcal{M} \models \neg\varphi$. By Lemma 90, there is an open branch in the tableau for $\neg\varphi$. Thus $\nvdash \varphi$. □

4.3. Exercises

EXERCISE 92. Suppose we are have a language $\mathcal{L}$ with two single place relation symbols P and Q; and one constant symbol. Write out the sentences from the first 3 stages of $Stage_{WFF+}$ noting which label each sentence comes from.

EXERCISE 93. In your own words:

(1) Explain *what* the completeness theorem says.
(2) Explain *why* is interesting and important.
(3) Explain *how* we proved it.

EXERCISE 94. Finish the rest of the cases from the proof Lemma 86.

EXERCISE 95. Complete the rest of the cases in Lemma 89.

EXERCISE 96. Use the Completeness Theorem to prove Lemma 75.

CHAPTER 5

Completeness 2

Goals:

- Prove the soundness of the natural deduction proof system.
- Prove the completeness of the natural deduction proof system.

THEOREM 97. $\Gamma \vdash_{Nat} \varphi$ *iff* $\Gamma \models \varphi$. [1]

The ($\rightarrow$) direction is known as *soundness*; and the ($\leftarrow$) direction is known as completeness.

This theorem is know as *strong completeness*, since we are permitted to use a set of sentences Γ from which to derive φ.

5.1. Soundness

5.1.1. Strategy. In a nutshell, soundness says that if we can derive φ from assumptions in Γ, then φ is true in all models where all of Γ is true. We are trying to show that when we construct a proof, the rules of our natural deduction system don't get us into any trouble: they don't derive things that aren't consequences.

Now given that we construct proofs in a stage-by-stage kind of fashion, this suggests that we might use induction to prove this theorem. We want to demonstrate a fact about all derivations so we might figure out a way of defining the derivations in a stage-by-stage way and then use our induction principle to complete the proof.

Of course, the notion of complexity/stage will not be the complexity of a formulae. We need something different. But the required notion is not difficult to think up. We just need to think about how a derivation is constructed using the rules of our system.

5.1.2. Proof.

[1] We shall omit the $_{Nat}$ for the remainder of this week's notes.

5.1.2.1. *Building derivations in stages .* Recall that we write $\Gamma \vdash_{ND} \varphi$ to mean that there is a (natural deduction) derivation of φ from (finitely many) assumptions contained in Γ. We want to describe a way of defining all of these derivations.

For this kind of definition to work, we need to things:

(1) a place/set from which to *start* (our base);
(2) a set of *rules* which tell use how to get from one stage to the next (our inductive jump).

Now it's easy to see what to use for the rules (2.). We are just going to use the rules of the natural deduction system.

For example, suppose we had derivations of $\Gamma \vdash \varphi$ and $\Delta \vdash \psi$. Then using the ($\wedge$-I) rule we can clearly get a derivation of $\Gamma, \Delta \vdash \varphi \wedge \psi$. All we are saying here is that we may put together the derivation d_φ of $\Gamma \vdash \varphi$ and the derivation of $d_\psi \vdash \psi$ such that one is above the other. Then the final lines of each derivation can be put together using ($\wedge$-I) to get a derivation of $\varphi \wedge \psi$ from assumptions in Γ, Δ.[2] Write this out in the Lemon style to convince yourself of this fact and to get more familiar with the notation.

Similarly, given a derivation of $\Gamma, \varphi \vdash \chi$ and a derivation of $\Gamma, \psi \vdash \chi$, we may use the ($\vee$-E) rule to construct a derivation of $\Gamma, \varphi \vee \psi \vdash \chi$. Again, write out the form of this derivation to convince yourself this is correct.

This tells us how we may move from one stage of our construction to another, but what is our base or atomic case. Again the obvious things works. The simplest derivation is simply the derivation of φ from the assumption φ itself. This is the first line of any derivation. We may write this as $\{\varphi\} \vdash \varphi$.

Now let us formally define a *derivation* in stages as follows. We let $Stage_{Der}(n)$ give us the n^{th} stage of our construction:

- $Stage_{Der}(1)$ is the set of all derivations of the form $\{\varphi\} \vdash \varphi$ for some sentence φ of our language; and
- $Stage_{Der}(n+1)$ is the set of all derivations which are either:
  - in $Stage_{Der}(n)$; or
  - can be constructed from derivations $d \in Stage_{Der}(n)$ using the rules of the natural deduction system.

[2] Strictly, since Γ and Δ are sets of sentences, we should write $\Gamma \cup \Delta$ rather than Γ, Δ, but we shall write it this way for convenience. No confusion should arise.

We then say that d is a *derivation* if there is some n such that $d \in Stage_{Der}(n)$. Alternatively and equivalently (as we saw in Week 1), we may define a derivation as follows:

- if d is of the form $\{\varphi\} \vdash \varphi$, then d is a derivation;
- if d_φ is a derivation of $\Gamma \vdash \varphi$ and d_ψ is a derivation of $\Delta \vdash \psi$, then the derivation d formed by combining d_φ and d_ψ using ($\wedge$-I) is a derivation of $\Gamma, \Delta \vdash \varphi \wedge \psi$;
- ... **add a case for each rule for the construction of derivations (see Exercise 110) ...**
- nothing else is a derivation.

With this method of building up derivations using rules we are ready to use the induction principle to complete a proof about all derivations and how they preserve truth.

5.1.2.2. *Showing that the rules preserve truth.* We first note the following simple (but important fact) which will be helpful in the proof and also significant.

PROPOSITION 98. *(Monotonicity) If $\Gamma \vdash \varphi$ and $\Delta \supseteq \Gamma$, then $\Delta \vdash \varphi$. Moreover, if d is a derivation witnessing that $\Gamma \vdash \varphi$, then d also witnesses that $\Delta \vdash \varphi$.*

So our goal theorem is the following:

THEOREM 99. *If $\Gamma \vdash \varphi$, then $\Gamma \models \varphi$.*

But it will be useful to unpack the definition of the consequent. Thus, we want to show that if there is a derivation such that $\Gamma \vdash \varphi$, then in every model $\mathcal{M}$ where every $\gamma \in \Gamma$ is true in $\mathcal{M}$, then so is φ.

PROOF. We proceed by induction on the complexity of derivations. Thus we show that "atomic" derivations uphold our goal fact and that constructions of new derivations based on the rules also uphold our goal fact.

(Base) Suppose d is a derivation of the form $\{\varphi\} \vdash \varphi$. Then we need to show that in every model $\mathcal{M}$ where every $\gamma \in \{\varphi\}$ is such that $\mathcal{M} \models \gamma$, we have $\mathcal{M} \models \varphi$. Let $\mathcal{M}$ be an arbitrary model in which every element of $\{\varphi\}$ is true. Thus $\mathcal{M} \models \varphi$; and this is exactly what we want.

(Induction step) Now suppose that we have shown that for every stage $m \leq n$, that

- if some derivation witnessing that $\Delta \vdash \psi$ is in $Stage_{Der}(m)$, then $\Delta \models \psi$.

This is our induction hypothesis. To complete the induction step, we need to show that the same is true for stage $n+1$. Let d which is in $Stage_{Der}(n+1) \backslash Stage_{Der}(n)$ (i.e., this is the first stage of the construction in which d is present - unlike say an atomic derivation which will be present in every level). Then by the definition of the stages, d must have been formed from derivation(s) in previous levels by one of the rules of the natural deduction system. So it will suffice if we show the result holds for each of the rules.

We shall do a few of the cases here and leave the rest as exercises.

($\wedge$-I) Suppose d_φ is a derivation of $\Gamma \vdash \varphi$ and d_ψ is a derivation of $\Delta \vdash \psi$ with both d_φ and d_ψ being from some stage n. Then the ($\wedge$-I)-rule says that the $n+1^{th}$ stage will consist of a derivation d such that $\Gamma, \Delta \vdash \varphi \wedge \psi$. To complete this case, we must show that $\Gamma, \Delta \models \varphi \wedge \psi$.

First observe that by Proposition 98, d_φ and d_ψ are also derivations witnessing that $\Gamma, \Delta \vdash \varphi$ and $\Gamma, \Delta \vdash \psi$. Then by induction hypothesis, since d_φ, d_ψ are less complex, we have $\Gamma, \Delta \models \varphi$ and $\Gamma, \Delta \models \psi$. Let $\mathcal{M}$ be a model in which $\mathcal{M} \models \gamma$ for all $\gamma \in \Gamma \cup \Delta$. Thus we have $\mathcal{M} \models \varphi$ and $\mathcal{M} \models \psi$, so $\mathcal{M} \models \varphi \wedge \psi$, which is what we wanted.[3]

($\forall$-I) Suppose we have a derivation d_φ of $\Gamma \vdash \varphi(a)$, where a does not occur in Γ or $\forall x \varphi(x)$ where d_φ is a member of stage n. Then by the rules of stage construction, there is a derivation d of $\Gamma \vdash \forall x \varphi(x)$ in stage $n+1$. We must show that $\Gamma \models \forall x \varphi(x)$.

By induction hypothesis, we see that $\Gamma \models \varphi(a)$. Let $\mathcal{M}$ be a model in which all of Γ is true. Then $\mathcal{M} \models \varphi(a)$. Suppose that $m = a^{\mathcal{M}}$. Now consider any model $\mathcal{M}'$ which is exactly like $\mathcal{M}$, except that $a^{\mathcal{M}'} = m' \neq a^{\mathcal{M}} = m$; i.e., we let $a^{\mathcal{M}'}$ be some arbitrary other m' from the domain M. Since Γ does not contain any sentence with the constant symbol a in it, it is *obvious* that $\mathcal{M}'$ is still a model of all of Γ.[4] Thus $\mathcal{M}' \models \varphi(a)$.

But the fact that all models $\mathcal{M}'$ whose only difference from $\mathcal{M}$ is their interpretation of the symbol a are such that $\mathcal{M}' \models \varphi(a)$ just means that:

$$\forall m \in M \ \mathcal{M}^+ \models \varphi(m) \quad \Leftrightarrow \quad \mathcal{M} \models \forall x \varphi(x).$$

[3] Observe that I am using both interpretations of the $\models$ symbol here. One represents the *satisfaction* reading and the other is giving us the *consequence* reading.

[4] Why is it clear? See Exercise 111.

($\vee$-E) This one is a little more complicated since we have more *plugging-in* involved in ($\vee$-E). Suppose:

- d_φ is a derivation of $\Gamma, \varphi \vdash \chi$;
- d_ψ is a derivation of $\Delta, \psi \vdash \chi$; and
- $d_{\varphi\vee\psi}$ is a derivation of $\Xi \vdash \varphi \vee \psi$.

which are all members of stage n. Then, the $(\vee - E)$ rule tells us that the $n + 1^{th}$ stage contains a derivation d of $\Gamma, \Delta, \Xi \vdash \chi$. We must show that that $\Gamma, \Delta, \Xi \models \chi$.

It will actually suffice to show that $\Gamma, \Delta, \varphi \vee \psi \models \chi$ since if $\mathcal{M} \models \gamma$ for all $\gamma \in \Gamma \cup \Delta \cup \Xi$, then $\mathcal{M} \models \gamma$ for all $\gamma \in \Xi$. Then since $d_{\varphi\vee\psi}$ is a derivation of less complexity (i.e., a member of the n^{th} stage) we see that $\Xi \models \varphi \vee \psi$. Thus $\mathcal{M} \models \varphi \vee \psi$ and $\mathcal{M} \models \chi$, which is what we need to show.

So let us suppose that $\mathcal{M}$ is such that for all $\gamma \in \Gamma \cup \Delta \cup \{\varphi \vee \psi\}$, $\mathcal{M} \models \gamma$. Then by induction hypothesis , we have both:

(1) $\Gamma, \varphi \models \chi$; and
(2) $\Delta, \psi \models \chi$.

Now since $\mathcal{M} \models \varphi \vee \psi$, either $\mathcal{M} \models \varphi$ or $\mathcal{M} \models \psi$. Suppose the former, then by (1.), $\mathcal{M} \models \chi$. Suppose the latter, then by (2.), $\mathcal{M} \models \chi$. Thus $\mathcal{M} \models \chi$, which is what we needed to prove.[5]

□

REMARK 100. Note how this move relies on the assumption that a is not free in $\forall x\varphi(x)$. If it had been, it would end up being bound by the universal quantifier in the move we make here and we wouldn't really have $\forall x\varphi(x)$ in such a case. For example, if we'd let $\varphi(x)$ be $x = a$, then we would have $\models \varphi(a)$; i.e., every $\mathcal{M}$ is such that $\mathcal{M} \models a = a$. Thus every $\mathcal{M}'$ where $a^{\mathcal{M}'} \neq a^{\mathcal{M}}$ is such that $\mathcal{M} \models a = a$. However, this only tells us that

$$\forall m \in M^+ \; \mathcal{M} \models m = m \;\; \Leftrightarrow \;\; \mathcal{M} \models \forall x(x = x).$$

Thus we get the universalisation of $x = x$, this is the wrong formula.

[5]Observe that we actually employed a version of the rule ($\vee$-E) in the metalanguage of our proof, when took up the two suppositions: first, that φ is true in $\mathcal{M}$; and second, that ψ is true in $\mathcal{M}$. This almost *circular* feature is a characteristic of soundness proofs.

5.2. Completeness

5.2.1. Strategy. At a high level, we are taking the fact that there is no proof of φ from assumptions in Γ and using that fact to make a set of sentences which describes a model in which $\Gamma \cup \{\neg\varphi\}$ is true.

As in the previous week, our strategy can be described as follows:

(1) Find a *special set of sentences* Δ to describe the model.
(2) Define a *model* $\mathcal{M}^\Delta$ using the special set; and
(3) Show that every member of $\Gamma \cup \{\neg\varphi\}$ is indeed *true* in $\mathcal{M}^\Delta$.

This will clearly suffice to establish our completeness theorem.

At this level, the proof has the *same* strategy as for the tableau completeness theorem. But as we get into the details there is less similarity.

5.2.2. The Proof.

5.2.2.1. *The special set.* In the tableau case, we had an obvious way of construction a special set: we used the sentences from an open branch. With the natural deduction system, there is no obvious counterpart. We do not have a canonical means of generating counterexamples with natural deduction: the system only gives us a means of proving things.

So what can we do? This is our position:

(1) We want to describe a model in which every sentence in $\Gamma \cup \{\neg\varphi\}$; and
(2) We know that $\Gamma \cup \{\neg\varphi\}$ is consistent.

Now we have no reason to think that $\Gamma \cup \{\neg\varphi\}$ contains enough information to describe a model, but what if we added more information. Perhaps we could keep on adding more sentences until we did have such a description. This will be our strategy.

So which sentences should we add? Let's make it simple and everything that we *can*. By "can" let us mean we add any sentence that is *consistent* with $\Gamma \cup \{\neg\varphi\}$. If, on the other hand, we added a sentence ψ to the set which was inconsistent with $\Gamma \cup \{\neg\varphi\}$, then we would know (by soundness) that there was no model of $\Gamma \cup \{\neg\varphi, \psi\}$. So we certainly don't want that.

We have no reason to think that adding sentence which are consistent will stop us from being able to find a model; indeed, by adding sentences which

are consistent with $\Gamma \cup \{\neg\varphi\}$, we hope to be able to pin down one particular model.

Intuitively speaking our process will be to start with the set $\Gamma \cup \{\neg\varphi\}$ and add as many sentences consistent with it as we can. Such a set will be called maximal consistent.

DEFINITION 101. A set of sentences Δ is *maximal consistent* if:

- Δ is consistent; and
- if $\varphi \notin \Delta$, then $\Delta \cup \{\varphi\}$ is not consistent.

So if we added a single extra sentence to a maximal consistent Δ, then the result would no longer be consistent.

Maximal consistent sets have a very useful property. They give us *(negation)-complete* sets of sentences.[6]

DEFINITION 102. A set of sentences Δ is *(negation)-complete* if for every sentence φ either $\varphi \in \Delta$ or $(\neg\varphi) \notin \Delta$.

A complete set of sentences Δ is going to be useful for describing a model, just as with a model $\mathcal{M}$ we have $\mathcal{M} \models \varphi$ or $\mathcal{M} \models \neg\varphi$ for all sentences φ we have $\varphi \in \Delta$ or $(\neg\Delta)$. This bivalence-like property gives us, intuitively speaking, the sufficiency of information to define the model.

THEOREM 103. *If Δ is maximal consistent, then Δ is complete.*

PROOF. Suppose Δ is maximal consistent. Let φ be an arbitrary sentence. to show that Δ is complete, it will suffice to show that either $\varphi \in \Delta$ or $(\neg\varphi) \in \Delta$. We first establish a helpful claim.

CLAIM. (i) If $\varphi \notin \Delta$, then $\Delta \vdash \neg\varphi$; (ii) If $(\neg\varphi) \notin \Delta$, then $\Delta \vdash \varphi$.

PROOF. (i) By maximality $\Delta \cup \{\varphi\}$ is not consistent. From assumptions in $\Delta \cup \{\varphi\}$, we can derive $\bot$. Since Δ is consistent, it is clear that φ must be one of the assumptions in this proof. By ($\neg$-I), we may thus get a proof of $\neg\varphi$ from Δ; i.e., $\Delta \vdash \neg\varphi$. (ii) similar. □

Now we suppose for *reductio*, that both φ and $\neg\varphi$ are not in Δ. By the claim above, we have $\Delta \vdash \neg\varphi$ and $\Delta \vdash \varphi$. Thus using ($\neg$-E), we may construct a proof of $\bot$ from Δ: contradicting our assumption that Δ is consistent. □

[6] Note the sense of *complete* here is different to that meant when we speak of the completeness of a proof system.

So this is a good start for our special set of sentences Δ. But there is something else we need. In a similar fashion to the completeness proof for tableau, we are going to use the constant symbols occurring in sentences of the set to form our domain.

Now suppose we have the following situation. For some sentence $\varphi := \exists x\psi(x)$ we have:

- $\exists x\psi(x)$ is in Δ; but
- for all $a \in M^{\Delta}$ (our domain of constant symbols) $(\neg\psi(a)) \in \Delta$.

There's something clearly wrong with this. If it's truth that there is a ψ, then something from the domain had better be an a.

Now if we only restrict ourselves to consistent extensions of $\Gamma \cup \{\neg\varphi\}$, then we cannot avoid this problem. To see this observe that there is no way of deriving $\bot$ from the set consisting of:

- $\exists x\psi(x)$; and
- $\neg\psi(a)$ for all $a \in M^{\Delta}$.

We'll simply never get into a position where the consistency constraint could (in general) block the problem.

The simple condition is to add a further condition on the set Δ. We demand that every existential sentence gets a *witness*; or more formally:

- if $\exists x\psi(x) \in \Delta$, then for some constant symbol a, $\psi(a) \in \Delta$.

We shall say that Δ is *existentially witnessed* if it enjoys this property.

We shall see that these two requirements are sufficient for our result. We now know what we require of our special set Δ, but how do we get such a set. Again using the techniques of Week 1, we are going to build up this set by recursion in stages.

We first take an enumeration $(\varphi_n)_{n\in\omega} = \varphi_1, \varphi_2, ..., \varphi_n, ...$ of all the sentences from $\mathcal{L}(C)$.

We now proceed as follows:

- Let $Stage_{\Delta}(\emptyset) = \Gamma \cup \{\neg\varphi\}$
- Let $Stage_{\Delta}(n+1)$ be the collection of sentences including:
  - sentences from $Stage_{\Delta}(n)$;
  - φ_{n+1} if φ_{n+1} is consistent with $Stage_{\Delta}(n)$; and

– $\psi(a)$ if $\varphi_{n+1} := \exists x\psi(x)$ where a is some constant symbol not in $Stage_\Delta(n)$ or φ_{n+1} and φ_{n+1} is consistent with $Stage_\Delta(n)$.

We then say that $\psi \in \Delta$ iff there is some n such that $\psi \in Stage_\Delta(n)$.

REMARK 104. Observe that we are always able to find a new constant symbol since there are infinitely many of them and at any stage of the construction we have only used finitely many.

Δ is then our special set. We then verify that Δ is indeed maximal consistent and is existentially witnessed.

LEMMA 105. *Δ is maximal consistent and existentially witnessed.*

REMARK 106. I use $\mathcal{D}$ to denote derivations in this proof (instead of d) in order to avoid confusion with constant symbols.

PROOF. Δ is existentially witnessed by the way we constructed the Δ.

If Δ is not consistent, then by LNP, there must have been some least stage at which $Stage_\Delta(n+1)$ became inconsistent while $Stage_\Delta(n)$ is consistent.

So clearly whatever was added to $Stage_\Delta(n)$ caused the inconsistency. There are two possible scenarios:

(i) $Stage_\Delta(n+1) = Stage_\Delta(n) \cup \{\varphi_{n+1}\}$ where φ_{n+1} is not of the form $\exists x\psi(x)$;

(ii) $Stage_\Delta(n+1) = Stage_\Delta(n) \cup \{\varphi_{n+1}, \psi(a)\}$ where φ_{n+1} is of the form $\exists x\psi(x)$.

In case (i), we see that by how we defined the stage construction that we only have

$$Stage_\Delta(n+1) = Stage_\Delta(n) \cup \{\varphi_{n+1}\}$$

when $Stage_\Delta(n) \cup \{\varphi_{n+1}\}$ is consistent. This means that $Stage_\Delta(n+1)$ is consistent after all.

In case (ii), we see that φ_{n+1} must be of the form $\exists x\psi(x)$. Supposing that

$$Stage_\Delta(n+1) = Stage_\Delta(n) \cup \{\varphi_{n+1}, \psi(a)\}$$

is inconsistent, there must be some derivation of $\perp$ from it. Consider such a derivation $\mathcal{D}$. We now show how to transform $\mathcal{D}$ into a derivation of $\perp$ from $Stage_\Delta(n) \cup \{\varphi_{n+1}\}$. By the same reasoning as for the case (i), this is a contradiction and will suffice for the Lemma.

Observe that a does not occur in $Stage_\Delta(n+1)$. Thus $\mathcal{D}$ cannot rely on an assumption in which a occurs free. But this just means that we can employ

($\exists$-E) on $\exists x\psi(x)$ and $\psi(a)$, thus discharging $\psi(a)$ as an assumption in $\mathcal{D}$. This leaves us with a derivation d' of $\Gamma, \exists x\psi(x) \vdash \bot$ which means that $\exists x\psi(x)$ was inconsistent with Γ after all. By the rules of the construction, this means that $\exists x\psi(x)$ could not be in $Stage_\Delta(n)$ which contradicts our assumption. $\square$

5.2.2.2. *Defining the model.* We now use Δ to define a model. We let the language be $\mathcal{L}(C)$.

Let $\mathcal{M}^\Delta$ be such that:

- M^Δ(the domain) is the set of constant symbols a occurring in sentences in Δ;
- for each constant symbol a, let $a^\mathcal{M} = a$ (i.e., itself);
- for each n-ary relation symbol R of $\mathcal{L}(C)$, let $R^\mathcal{M}$ be the set of n-tuples $\langle a_1, ..., a_n\rangle$ such that $Ra_1, ..., a_n$ is in Δ.

5.2.2.3. *Showing that $\mathcal{M}^\Delta \models \Gamma \cup \{\neg\varphi\}$.* As with the tableau proof, we now want to show that every sentence in $\Gamma \cup \{\neg\varphi\}$ is in fact true in $\mathcal{M}^\Delta$.

Once again we show something stronger.

THEOREM 107. *$\varphi \in \Delta$ iff $\mathcal{M}^\Delta \models \varphi$.*

REMARK 108. Observe that we have an "iff" here, as opposed to the "if ..., then ..." from the tableau completeness proof.

Again our strategy here is going to be a proof by induction. However, in contrast to the completeness proof for tableau, this is much simpler. We can get away with using our simple definition of the complexity of formulae (rather that $+$-complexity or something even more exotic).

The following claim is very helpful.

CLAIM 109. Suppose Δ is maximal consistent. Then $\Delta \vdash \psi$ iff $\psi \in \Delta$.

PROOF. ($\rightarrow$) Suppose $\Delta \vdash \psi$. For *reductio* suppose $\psi \notin \Delta$. Then by Lemma 103 we $(\neg\psi) \in \Delta$. Then we could make a derivation of $\bot$ from sentences in Δ: contradicting the consistency of Δ. Thus $\psi \in \Delta$.

($\leftarrow$) Suppose $\psi \in \Delta$. Then we may trivially derive ψ from $\{\psi\} \subseteq \Delta$. $\square$

PROOF. (of Theorem 107) We proceed by induction on the complexity of formulae.

(Base) Exercise.

(Induction Step) Suppose that for all sentence of complexity $\leq n$, we have shown that $\psi \in \Delta$ iff $\mathcal{M}^{\Delta} \models \psi$. We show that the same is true for formulae of complexity $n+1$.

Suppose $\psi := \chi \wedge \delta \in \Delta$. We claim that $\chi \wedge \delta \in \Delta$ iff both $\chi \in \Delta$ and $\delta \in \Delta$. If $\chi \wedge \delta \in \Delta$, then by Claim 109 and ($\wedge$-E) we have both χ and δ in Δ. If both χ and δ are in Δ then Claim 109 and ($\wedge$-I) tell us that $\chi \wedge \delta \in \Delta$.

From here we have:

$$\begin{aligned} \chi \wedge \delta \in \Delta &\Leftrightarrow \chi \in \Delta \ \& \ \delta \in \Delta \\ &\Leftrightarrow \mathcal{M}^{\Delta} \models \chi \ \& \ \mathcal{M}^{\Delta} \models \delta \\ &\Leftrightarrow \mathcal{M}^{\Delta} \models \chi \wedge \delta. \end{aligned}$$

The first $\Leftrightarrow$ is via our claim; the second $\Leftrightarrow$ is by induction hypothesis; and the last $\Leftrightarrow$ is from the $\models$ definition.

Suppose $\psi := \neg\chi$. Then

$$\begin{aligned} (\neg\chi) \in \Delta &\Leftrightarrow \chi \notin \Delta \\ &\Leftrightarrow \mathcal{M}^{\Delta} \not\models \chi \\ &\Leftrightarrow \mathcal{M}^{\Delta} \models \neg\chi. \end{aligned}$$

The first $\Leftrightarrow$ is via the completeness of Δ; the second $\Leftrightarrow$ is by induction hypothesis; and the last is via the $\models$ definition.

Suppose $\psi := \exists x \chi(x)$. Then we claim that $\exists x \chi(x) \in \Delta$ iff there is some constant symbol a in $\mathcal{L}(C)$ such that $\chi(a) \in \Delta$. Suppose $\exists x \chi(x) \in \Delta$. Then by construction of Δ (i.e., since it is existentially witnessed) there is some $a \in \mathcal{L}(C)$ such that $\chi(a) \in \Delta$. On the other hand if for some $a \in \mathcal{L}(C)$, $\chi(a) \in \Delta$, then by Claim 109 and ($\exists$-I), $\exists x \chi(x) \in \Delta$. Then we have

$$\begin{aligned} \exists x \chi(x) \in \Delta &\Leftrightarrow \text{there is some } a \in M^{\Delta} \text{ such that } \chi(a) \in \Delta \\ &\Leftrightarrow \text{there is some } a \in M^{\Delta} \text{ such that } \mathcal{M}^{\Delta} \models \chi(a) \\ &\Leftrightarrow \mathcal{M}^{\Delta} \models \exists x \chi(x). \end{aligned}$$

The first $\Leftrightarrow$ came from our claim; the second $\Leftrightarrow$ was via the induction hypothesis; and the final $\Leftrightarrow$ was via the $\models$ definition.

The other cases are left as exercises. $\square$

5.3. Exercises

EXERCISE 110. In Section 5.1.2.1, our second definition has not been completed. Complete the definition by adding the appropriate clauses for each of the other rules in the system.

EXERCISE 111. In the proof of Theorem 99, we make a claim about something being obvious (there is a footnote highlighting this). Establish the claim.

EXERCISE 112. Complete the rest of the cases in the proof of 99.

EXERCISE 113. Complete part (ii) of the claim in Theorem 103.

EXERCISE 114. Explain why there is no derivation of $\perp$ from the set consisting of:

- $\exists x \psi(x)$; and
- $\neg\psi(a)$ for all $a \in M^{\Delta}$.

EXERCISE 115. In your own words:

(1) Explain *what* the completeness theorem says.
(2) Explain *why* is interesting and important.
(3) Explain *how* we proved it.

EXERCISE 116. Complete the base case for Theorem 107.

EXERCISE 117. Complete the rest of the cases for the proof of Theorem 107.

CHAPTER 6

Model Theory

Goals:

- Look at some basic model theory.
- Prove the compactness theorem.

6.1. Isomorphism and Elementary Equivalence

In this week, we are going to be concerned with relationships between models.

Consider two models $\mathcal{M}$ and $\mathcal{N}$ of some language $\mathcal{L}$. All that they have in common is the fact that they have interpretations for the same relation, constant and function symbols.

We now consider two ways in which $\mathcal{M}$ and $\mathcal{N}$ may be very alike: isomorphism and elementary equivalence. Before we do this we'll need a few simple concepts from set theory.

A *map* is a function, which we may denote as say σ, which has a domain say D and a codomain C. It takes any object in D and *outputs* an object from C. We abbreviate this by writing $\sigma : D \to C$.[1]

We say that a map $\sigma : D \to C$ is *surjective* if for every element $c \in C$, there is some $d \in D$ such that $\sigma(d) = c$. In other words, the map σ *exhausts* its codomain. (Sometimes such maps are called *onto*.)

We say that a map $\sigma : D \to C$ is *injective* if for any two elements $d_1 \neq d_2 \in D$, $\sigma(d_1) \neq \sigma(d_2)$. Thus different object in the domain are always output to different object in the codomain. (Sometimes such maps are called *one-to-one*.)

We say that a map is *bijective* if it is both surjective and injective.

[1] Note that there is no requirement that every object in C the result of σ's application to some object $d \in D$. Some $c \in C$ may be missed, as it were.

DEFINITION 118. Let us say that models $\mathcal{M}$ and $\mathcal{N}$ are *isomorphic*, abbreviated $\mathcal{M} \cong \mathcal{N}$, if there is a map σ between M and N (their respective domains) such that:

- σ is bijective;
- for every constant symbol c from $\mathcal{L}$, we have
$$\sigma(c^{\mathcal{M}}) = c^{\mathcal{N}};$$
- for every n-ary relation symbol R in $\mathcal{L}$ and any $m_1, ..., m_n$ from M, we have
$$\langle m_1, ..., m_n \rangle \in R^{\mathcal{M}} \Leftrightarrow \langle \sigma(m_1), ..., \sigma(m_n) \rangle \in R^{\mathcal{N}}; \text{ and}$$
- for every n-ary function symbol f in $\mathcal{L}$ and any $m_1, ..., m_n$ from M, we have
$$\sigma(f^{\mathcal{M}}(m_1, ..., m_n)) = f^{\mathcal{M}}(\sigma(m_1), ..., \sigma(m_2)).$$

We shall write $\sigma : \mathcal{M} \cong \mathcal{N}$ to indicate that σ is the map described above which witnesses the isomorphism.

REMARK 119. There is a sense in which isomorphic models are basically the same. The only thing that distinguishes two isomorphic models is the material from which they are constructed. However, from the point of view of the language $\mathcal{L}$ and what we can express in it, this kind of difference is invisible, or perhaps better, ineffable.

Elementary equivalence is also a relation between models which expresses the fact that they are alike in some way.

DEFINITION 120. Two models $\mathcal{M}$ and $\mathcal{N}$ of some language $\mathcal{L}$ are *elementary equivalent*, abbreviated $\mathcal{M} \equiv \mathcal{N}$, if for every sentence $\varphi \in Sent_{\mathcal{L}}$, we have:
$$\mathcal{M} \models \varphi \Leftrightarrow \mathcal{N} \models \varphi.$$

Thus two models are elementary equivalent, they make exactly the same sentences true.

We might wonder if elementary equivalence and isomorphism just mean the same thing. We shall see in the next section that they do not.

6.2. The compactness theorem

The compactness theorem is the foundation stone of basic model theory. It is the engine in the background of all of the basic theorem in this research area. However, it *perhaps* does not, at first glance, appear to be so significant. To appreciate it, we'll need to look at how it can be applied.

As usual we'll need a few definitions to get things moving.

We already say that a sentence φ of some $\mathcal{L}$ is *satisfiable* if there is some $\mathcal{M}$ such that $\mathcal{M} \models \varphi$.

Let Γ be a set of sentences from $\mathcal{L}$. We shall say that Γ is *satisfiable* if every sentence $\gamma \in \Gamma$ is such that $\mathcal{M} \models \gamma$; and we'll abbreviate this by $\mathcal{M} \models \Gamma$ (another overloading of the $\models$ symbol).

We shall say that Γ is *finitely satisfiable* if for every finite subset Δ of Γ, Δ is satisfiable; i.e., there is some $\mathcal{M}$ such that $\mathcal{M} \models \Delta$.

We may now state the compactness theorem.

THEOREM 121. *(Compactness) If Γ is finitely satisfiable, the Γ is satisfiable.*

In other words, if every finite subsets of Γ has a model, then so does Γ itself.

PROOF. We proceed by contraposition. Suppose that Γ is not satisfiable. This just means that $\Gamma \models \bot$; i.e., there is no model which makes all of Γ true. By completeness, we then see that $\Gamma \vdash \bot$. Thus there must be some finite set of assumptions Δ from which we may derive $\bot$ (in say the natural deduction system); i.e. $\Delta \vdash \bot$. But then, by soundness we get $\Delta \models \bot$; or in other works, there is no model $\mathcal{M}$ which makes all of Δ true. Thus Γ is not finitely satisfiable. □

6.2.1. Nonstandard models. We now show how elementary equivalence and isomorphism come apart.

Let $\mathbb{N}$ be the standard model of arithmetic as described in Example 34 from Week 2. For convenience, let's assume that we have a constant symbol for every natural number. It could just be our everyday Arabic representation of it. So strictly speaking we have expanded the language of arithmetic and its standard model to accommodate all these new constant symbols. We shall only take this approach in this section.

Another definition is helpful at this point.

DEFINITION 122. Given a model $\mathcal{M}$, let the *theory of* $\mathcal{M}$, $Th(\mathcal{M})$ be the set of sentences from $\mathcal{L}$ which are true in $\mathcal{M}$.

THEOREM 123. *There is a model* $\mathcal{M}$ *such that* $\mathcal{M} \models Th(\mathbb{N})$ *but* $\mathcal{M} \ncong \mathbb{N}$.

PROOF. Our strategy here will be to employ the compactness theorem. Let us expand $\mathcal{L}$ with the addition of a single new constant symbol c and call the result $\mathcal{L}_c$. Now consider $\Gamma \subseteq Sent_{\mathcal{L}_c}$ which consists of:

- all of the sentences in $Th(\mathbb{N})$; and
- the set of sentences of the form $c \neq \bar{n}$ where $\bar{n}$ is a constant symbol denoting n.

Now we claim that Γ is finitely satisfiable. Let Δ be some finite subset of Γ. To get a model $\mathcal{N}$ for Δ, we just expand the standard model $\mathbb{N}$ by giving an interpretation of the new constant symbol c. Since Δ is finite there must be some least n such that the sentence $c \neq n$ is not in Δ. Let $c^{\mathcal{N}} = n$ for that n. This is all that is required, so $\mathcal{N} \models \Delta$ and Γ is finitely satisfiable.

By compactness, there must be a model $\mathcal{M}$ such that $\mathcal{M} \models \Gamma$. Fix such an $\mathcal{M}$.

Now we claim that $\mathcal{M} \ncong \mathbb{N}$. Suppose that there was an isomorphic map σ between M and ω (ω is the set of natural numbers). σ will need to be such that every constant symbol a is such that σ maps $a^{\mathcal{M}}$ to $a^{\mathbb{N}}$. For the Arabic numbers there is only one thing we can do. We have

$$\begin{aligned} \sigma(0^{\mathcal{M}}) &= 0^{\mathbb{N}} \\ \sigma(1^{\mathcal{M}}) &= 1^{\mathbb{N}} \\ &\vdots \\ \sigma(n^{\mathcal{M}}) &= n^{\mathbb{N}} \\ &\vdots \ . \end{aligned}$$

But since $\mathcal{M}$ is such that for all n, $\mathcal{M} \models c \neq n$, there is no place for σ to map $c^{\mathcal{M}}$to in $\mathbb{N}$. Thus σ fails the condition on constant symbols and $\mathcal{M} \ncong \mathbb{N}$. □

COROLLARY 124. *It is not the case that if* $\mathcal{M} \equiv \mathcal{N}$, *then* $\mathcal{M} \cong \mathcal{N}$.

6.3. Submodels & Embeddings

We now consider some relationships between models which involve less similarity. For simplicity, we shall only consider a language with function symbols.

DEFINITION 125. Let $\mathcal{M} = \langle M, c^{\mathcal{M}}, ..., R^{\mathcal{M}}, ...\rangle$ be a model of the language $\mathcal{L} = \{c, ..., R, ...\}$. $\mathcal{N} = \langle N, c^{\mathcal{N}}, ..., R^{\mathcal{N}}, ...\rangle$ is a *submodel* of $\mathcal{M}$, abbreviated $\mathcal{N} \subseteq \mathcal{M}$ if:

- $N \subseteq M$ (i.e., the domain of N is a subset of M);
- for every constant symbol c in $\mathcal{L}$, $c^{\mathcal{M}} = c^{\mathcal{N}}$; and
- for every n-ary relation symbol R in $\mathcal{L}$, $R^{\mathcal{N}}$ is the set of n-tuples $\langle m_1, ..., m_n\rangle$ where $m_1, ..., m_n \in N$ and $\langle m_1, ..., m_n\rangle \in R^{\mathcal{M}}$.

Essentially, to make a submodel of $\mathcal{M}$, we just take objects away from the domain M of $\mathcal{M}$.

We now consider the *embedding* relationship.

DEFINITION 126. Let $\mathcal{N}$ and $\mathcal{M}$ be models of some language $\mathcal{L}$. We say that f *embeds* $\mathcal{N}$ *into* $\mathcal{M}$, if there is some $\mathcal{M}' \subseteq \mathcal{M}$ such that $f : \mathcal{N} \cong \mathcal{M}'$; i.e., $\mathcal{N}$ is isomorphic to a submodel of $\mathcal{M}$.

6.3.1. Putting these notions to work. We now ask ourselves about the impact of these relations: what does it mean for $\mathcal{M}$ to be a submodel of $\mathcal{N}$?

Well one thing we might want to know is what sort of things are true in both $\mathcal{M}$ and $\mathcal{N}$.

Suppose $\mathcal{M}$ and $\mathcal{N}$ are models of the language $\mathcal{L} = \{P, Q\}$ which we describe as follows:

- $M = \{a, b\}$;
- $P^{\mathcal{M}} = \emptyset$ (i.e., $P^{\mathcal{M}} = \{\}$);
- $Q^{\mathcal{M}} = \{a, b\}$;
- $N = \{a, b, c\}$;
- $P^{\mathcal{N}} = \{a\}$; and
- $Q^{\mathcal{N}} = \{a, b\}$;

Clearly $\mathcal{M} \subseteq \mathcal{N}$.

Consider the sentence $\exists x Px$. Clearly this sentence is true in $\mathcal{N}$: there is some $n \in N$ (i.e., a) such that $n \in P^{\mathcal{N}}$. However, it is not true in $\mathcal{M}$.

Now consider the sentence $\forall x Qx$. It is true in $\mathcal{M}$ since everything in M is in $Q^{\mathcal{M}}$. On the other hand, $\mathcal{N} \not\models \forall x Qx$ since $c \notin Q^{\mathcal{N}}$.

So the sentence beginning with $\exists$ was made false when we went *down* to a submodel; and the sentence beginning with $\forall$ was made false when we went *up* to a supermodel.

What about the other direction? What would happen to a $\exists$ sentence if we went *up* to a supermodel? What would happen to a $\forall$ sentence if we went *down* to a submodel?

To explore this, we give a formal definition of the kinds of sentences we are interested in here.

DEFINITION 127. Let us say that a sentence φ is Σ_1 (a $\exists$-sentence) if φ is of the form $\exists x \psi(x)$ where $\psi(x)$ is a formula with (at most) one free variable, x, *containing no quantifiers*. Let us say that a sentence φ is Π_1 (a $\forall$-sentence) if φ is of the form $\forall x \psi(x)$ where $\psi(x)$ is a formula with at most one free variable, x, *containing no quantifiers*.

THEOREM 128. *Suppose$\mathcal{M} \subseteq \mathcal{N}$. Then,*

(1) *If φ is Σ_1 and $\mathcal{M} \models \varphi$, then $\mathcal{N} \models \varphi$;*
(2) *If φ is Π_1 and $\mathcal{N} \models \varphi$, then $\mathcal{M} \models \varphi$.*

To get this moving we need the following lemma.

LEMMA 129. *Let $\psi(x)$ be a formula without quantifiers with one free variable and let $\mathcal{M} \subseteq \mathcal{N}$. Then let $\mathcal{L}(M)$ be the expansion of $\mathcal{L}$ with constant symbols for every member of M and $\mathcal{M}^+$ and $\mathcal{N}^\dagger$be the appropriate expansions of $\mathcal{M}$ and $\mathcal{N}$. Then for all $m \in M$ we have*

$$\mathcal{M}^+ \models \psi(m) \;\Leftrightarrow\; \mathcal{N}^\dagger \models \psi(m).$$

[2]

PROOF. It should be clear that the formulae of $\mathcal{L}(M)$, which do not involve quantification can be build up inductively in stages by simply omitting the rules for the quantifiers. We thus proceed by induction on the complexity of formulae in $\mathcal{L}(M)$. It should also be clear that $\mathcal{M}^+ \subseteq \mathcal{N}^\dagger$.

[2] I've written $\mathcal{N}^\dagger$ to distinguish it from $\mathcal{N}^+$ which we defined in week to as the expansion of $\mathcal{N}$ which accommodates $\mathcal{L}(N)$.

(Base) Suppose that $\psi(x)$ is atomic. For example, let $\psi(x)$ be $Rxc_1...c_n$. Take an arbitrary $m \in M$. Then we have

$$\begin{aligned} \mathcal{M}^+ \models Rmc_1...c_n &\Leftrightarrow \langle m^{\mathcal{M}}, c_1^{\mathcal{M}}, ..., c_n^{\mathcal{M}} \rangle \in R^{\mathcal{M}} \\ &\Leftrightarrow \langle m^{\mathcal{N}}, c_1^{\mathcal{N}}, ..., c_n^{\mathcal{N}} \rangle \in R^{\mathcal{N}} \\ &\Leftrightarrow \mathcal{N}^+ \models Rmc_1...c_n. \end{aligned}$$

The first and third $\Leftrightarrow$ are by the satisfaction definition; and the second $\Leftrightarrow$ follows from the fact that $\mathcal{M}^+ \subseteq \mathcal{N}^\dagger$.

(Induction step) To save space, we just work through the case for $\neg$ and $\wedge$.

Suppose $\psi(x)$ is of the form $\neg\chi(x)$. Then taking arbitrary $m \in M$, we have

$$\begin{aligned} \mathcal{M}^+ \models \neg\chi(m) &\Leftrightarrow \mathcal{M}^+ \not\models \chi(m) \\ &\Leftrightarrow \mathcal{N}^\dagger \not\models \chi(m) \\ &\Leftrightarrow \mathcal{N}^\dagger \models \neg\chi(m). \end{aligned}$$

We leave the explanation of the steps as an exercise.

Suppose $\psi(x)$ is of the form $\chi(x) \wedge \delta(x)$. Then taking an arbitrary $m \in M$, we have

$$\begin{aligned} \mathcal{M}^+ \models \chi(m) \wedge \delta(m) &\Leftrightarrow \mathcal{M}^+ \models \chi(m) \ \& \ \mathcal{M}^+ \models \delta(m) \\ &\Leftrightarrow \mathcal{N}^\dagger \models \chi(m) \ \& \ \mathcal{N}^\dagger \models \delta(m) \\ &\Leftrightarrow \mathcal{N}^\dagger \models \chi(m) \wedge \delta(m). \end{aligned}$$

We leave the explanation of the steps as an exercise. □

With this in hand the proof of the theorem is easy.

PROOF. (of Theorem 128) We let $\mathcal{L}(M)$ be the expansion of $\mathcal{L}$ with constants for every elements of M and $\mathcal{M}^+$and $\mathcal{N}^\dagger$ be expansion of the models $\mathcal{M}$ and $\mathcal{N}$ with those new constant symbols interpreted appropriately.

(1.) Suppose φ is of the form $\exists x\psi(x)$ and $\mathcal{M} \models \exists x\psi(x)$. Then there is some $m \in M$ such that $\mathcal{M}^+ \models \psi(m)$ and by Lemma 129, $\mathcal{N}^\dagger \models \psi(m)$ and so $\mathcal{N} \models \exists x\psi(x)$.

(2.) Suppose φ is of the form $\forall x\psi(x)$ and $\mathcal{N} \models \forall x\psi(x)$. Then for any $m \in M$, we have $\mathcal{N}^\dagger \models \psi(m)$ (we actually know something stronger than this, but this is sufficient). Thus by Lemma 129, we see that for all $m \in M$, $\mathcal{M}^+ \models \psi(m)$; or in other words, $\mathcal{M}^+ \models \forall x\psi(x)$. □

REMARK 130. So the moral of this story is that for very simple formulae,

- if we find a witness then that witness will still be in the bigger model; and
- if everything in some domain satisfies the formula then this will still be the case in a smaller model.

6.4. Basic Set Theory

Up until now we have been considering two kinds of models: finite models and infinite models.

The size of a model is known as its *cardinality*. So if the domain has five objects in it, then its cardinality is simply five. And so on for any other finite cardinality.

When we come to an infinite model, perhaps constructed from an open branch we might be tempted to just say that its cardinality is infinite.

But this presupposes that there is *just one* infinite cardinality.

In one of the bigger surprises in this history of mathematics, this assumption (despite how reasonable it sounds) turns out to be false. We shall demonstrate this and then examine its effect on our theory of models.

6.4.1. Cardinality. First we need to be more precise about what it means to have a certain cardinality.

DEFINITION 131. Let us say that two collections A and B have the *same cardinality*, which we abbreviate $A \approx B$ if there is some map $f : A \to B$ which is a bijection.

Thus if A is a collection of five sheep and B is a collection of five ducks, then there is a map f which takes each sheep to a duck such that:

(1) every duck d is such that there is a sheep s for which $f(s) = d$; and
(2) if $s_1 \neq s_2$ are different sheep then they are mapped to different ducks ($f(s_1) \neq f(s_2)$);

i.e., f is a bijection.

DEFINITION 132. We shall say that collection A has less than or the same cardinality as B, abbreviated $A \prec B$, if there is an injection between A and B.

Thus if in the previous example there had been six ducks, we still could have got a map with (2.) satisfied, but one duck would have to have been left out.

So we should be able to see that these definitions are in accord with our intuitions about finite sets, but what about infinite ones. The following fact should be obvious, but it is instructive to prove it from our definitions.

FACT 133. *Let E be the set of even numbers and O be the set of odd numbers. Then $E \approx O$.*

PROOF. To show this, it suffices to define a bijection between E and O. Intuitively, we let f take the first even number to the fist odd number, the second even number to the second odd number and so on. More formally,

$$f(2n) = 2n + 1.$$

It should be clear that this is a bijection. □

The following fact may be more surprising.

FACT 134. *Let E be the set of even numbers and ω be the set of natural numbers. Then $N \approx E$.*

PROOF. Again we define the bijection. Let $f : N \to E$ be such that

$$f(n) = 2n.$$

This is clearly a surjection: every even number is output. Moreover it is an injection. Suppose not. Then there would be some even e such that for some $m \neq n$,

$$f(m) = e = f(n).$$

But then $2m = 2n$ and $m = n$: contradiction. □

This might be surprising given that finite collections A and B is A is a proper subset of B, $A \subsetneq B$ (i.e., $A \subseteq B$ but $A \neq B$), then $A \prec B$. This is not always the case with infinite collections.

6.4.1.1. *Enumeration.* We shall say that a collection A is *countable* if $N \approx A$; i.e., if there is a bijection f between N (the natural numbers and A). We shall say that f *enumerates* A since that's exactly what it does.

6.4.2. Cantor's theorem - the are larger infinities. We are not ready to prove that there is more than one size of infinite collection. This is known as Cantor's theorem.

The proof doesn't take that long, but we'll go through it somewhat informally to make it as clear as possible.

Suppose we had countably many one pound coins (a pleasant thought) and we lined them all up in a row. Each coin would either be heads-up or tails-up. Remember that since we have countably many coins, this means we can enumerate them with a function from N to the coins. Let us call this function c.

We may then represent this situation in a table as follows:

$c(0)$	$c(1)$	$c(2)$	$c(3)$	$c(4)$	$c(5)$	$c(6)$	
H	H	T	H	T	H	H	...

Now of course there are different ways that the coins could have been laid out. For example we might switch the third coin $c(2)$ from tails to heads.

Let us then consider the table which would result by placing each different arrangement of the coins in new rows of the table. Thus we get something like:

$c(0)$	$c(1)$	$c(2)$	$c(3)$	$c(4)$	$c(5)$	$c(6)$	...
H	H	T	H	T	H	H	...
T	T	T	T	T	T	T	...
H	T	T	H	T	T	H	...
T	H	T	T	T	H	H	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

We won't worry about the order in which the rows are filled in. We just want to ensure that every (infinite) arrangement of heads and tails is represented on exactly one row of the table.

Now our claim is that there are more rows than there are columns. Since there are infinitely many columns, this will suffice to show that there is more than one size of infinite collection.

So what does it mean for there to be more rows than columns? Since we know that the columns are countable, it will suffice to show that the rows are not countable. In order to do this, we must show that there is no bijective function r from the naturals to the rows, which provides an enumeration of them.

We shall demonstrate this by *reductio*. Thus, suppose that there was such an enumeration. Let us call it r. We might then represent this situation as follows:

	$c(0)$	$c(1)$	$c(2)$	$c(3)$	$c(4)$	$c(5)$	$c(6)$	...
$r(0)$	H	H	T	H	T	H	H	...
$r(1)$	T	T	T	T	T	T	T	...
$r(2)$	H	T	T	H	T	T	H	...
$r(3)$	T	H	T	T	T	H	H	...
$r(4)$	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

To show that r cannot enumerate all of the rows, we are going to *construct* a row $r^\dagger$ that the enumeration r must miss. This is an arrangement of the coins.

We let $r^\dagger$ be the row constructed by flipping every coin down the diagonal of our table. So looking at the table above we get:

	$c(0)$	$c(1)$	$c(2)$	$c(3)$	$c(4)$	$c(5)$	$c(6)$	...
$r(0)$	**T**	H	T	H	T	H	H	...
$r(1)$	T	**H**	T	T	T	T	T	...
$r(2)$	H	T	**T**	H	T	T	H	...
$r(3)$	T	H	T	**H**	T	H	H	...
$r(4)$	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Or more formally, we let $r^\dagger$ be such that the n^{th} column of $r^\dagger$ is:

- heads if the n^{th} column of the n^{th} row is tails; and
- tails if the n^{th} column of the n^{th} row is heads.

Now if $r^\dagger$ was in the enumeration, then there would have to be some n such that $r(n) = r^\dagger$. But this is not possible. We have defined $r^\dagger$ so that it is different from every $r(n)$ at exactly one place. Thus there can be no enumeration of the rows and there is a infinite cardinality which is *not countable*. We call such sizes *uncountable*.

REMARK 135. More informally, we observe that if there had been the same number of rows as columns, then the table above would be an infinite square. We have shown that there must always be an extra row beyond this.

6.4.2.1. *This also tells us ...* Consider each of the rows of the table. There is a sense in which each of them represents a particular subset of the natural numbers.

For example,

	$c(0)$	$c(1)$	$c(2)$	$c(3)$	$c(4)$	$c(5)$	$c(6)$	
$r(0)$	**H**	**H**	T	**H**	T	**H**	**H**	...

the 0^{th} row could pick used to pick out the set

$$\{0, 1, 3, 5, 6, ...\}.$$

We simply take the set be numbers of those coins which are facing heads-up. Moreover it should be clear that every set of natural numbers will be represented by exactly one of these rows.

Thus we have shown:

COROLLARY 136. *There are more sets of natural numbers than there are natural numbers.*

Moreover, it is possible to represent any real number by an infinitely long decimal number. We might then replace the coins in the example by a 10-sided dice. Then each of the rows could represent a real number between 0 and 1. We can then perform much the same trick as before to show that:

FACT 137. *There are more real numbers than natural numbers.*

The sizes of sets does not stop here either. We can repeat a (slightly more general) version of our argument above with the coins to get a collection which is even larger than the sets of all sets of natural numbers. Indeed we can repeat this indefinitely.

A full discussion of this topic would open us up in to the world of set theory and the transfinite.

6.5. Löwenheim-Skolem Theorems

But we are going to return to our discussion of models and consider what impact cardinality has upon it. Our target question here is going to be:

- Given a theory T (i.e. a set of sentence) which has a model $\mathcal{M}$ will there be other models $\mathcal{N}$ which have different cardinalities?

The answer to this question comes in two theorems which show that in terms of cardinality, there is a massive amount of freedom here.

THEOREM 138. *(Downward Löwenheim-Skolem Theorem) Let T be a countable theory and $\mathcal{M}$ be a model of T of arbitrary cardinality. Then T has a countable model.*

PROOF. By soundness, since T has a model it is consistent. Then by the techniques of our completeness proofs we may construct a model which has a countable domain. □

THEOREM 139. *(Upward Löwenheim-Skolem Theorem) Let $\mathcal{M}$ be an infinite model such that $\mathcal{M} \models T$ and let A be a set such that $M \prec A$. Then there is some $\mathcal{N}$ such that $N \approx A$ and $\mathcal{N} \models T$.*

In other words, given a model $\mathcal{M}$ of some theory and any larger set A, we can find another model $\mathcal{N}$ of T which is the same size as A.

PROOF. Let us expand the language $\mathcal{L}$ with a constant symbol c_a for every element $a \in A$. Then we let S be the theory consisting of those sentences in T and in the set

$$\{c_a \neq c_b \mid a, b \in A \; a \neq b\}.$$

Clearly, every finite subset of S is satisfiable. Thus S is satisfiable in some model $\mathcal{N}^+$. Since every one of the constant symbols denotes a different element of N^+, we have $N^+ \approx A$. Let $\mathcal{N}$ be the reduct of $\mathcal{N}^+$ back to the language $\mathcal{L}$. Clearly $\mathcal{N} \models T$. □

6.6. Exercises

EXERCISE 140. Show that if $\mathcal{M} \cong \mathcal{N}$, then $\mathcal{M} \equiv \mathcal{N}$.

EXERCISE 141. If $\mathcal{M} \equiv \mathcal{N}$, what can you say about $Th(\mathcal{M})$ and $Th(\mathcal{N})$?

EXERCISE 142. In Definition 125, we avoided function symbols in our definition. What do you think would be the appropriate condition to place here? [Note that we want the submodel to be a genuine model and so functions must work properly. Ensuring this may involve changing other parts of the submodel definition.]

EXERCISE 143. Explain how each of the $\Leftrightarrow$'s are justified in Lemma 129.

EXERCISE 144. Show using counterexamples that Theorem 128 would have failed if we had used arbitrary sentences beginning with $\exists$ and $\forall$ respectively.

PROBLEM 145. A theory Γ is a set of sentences which is closed under the consequence relation; i.e., if $\Gamma \models \varphi$, then $\varphi \in \Gamma$. Suppose Γ and Δ are both theories. For each of the following statements, either prove it or refute it with a counterexample:

(1) $\{\varphi \mid \varphi \in \Gamma \vee \varphi \in \Delta\}$ is a theory; and
(2) $\{\varphi \mid \varphi \notin \Gamma\}$ is a theory.

PROBLEM 146. Let $\mathcal{L}$ be a language with a finite number of relation, function and constant symbols. Show that the set of well-formed formulae of this language is countable.

PROBLEM 147. It is possible to formulate a theory of sets in first order logic. In this theory, we can prove Cantor's theorem and thus show that there are uncountable collections of sets. However, since the theory is formulated in first order logic we can also show that it has a countable model. What is going on here?

Part 2

Recursion & Incompleteness

CHAPTER 7

Recursion theory 1

In the first half of the course, we almost ignored the case of functions whenever they were distracting. In this half of the course, functions are going to play the starring role.

Recursion theory is the study of functions which are *computable*. Informally speaking, these are the functions which we *compute* in the sense that given the inputs of the function there is some set of rules for the calculation/construction such that after a finite amount of time (and perhaps paper) we can figure out the answer.

You might have thought that every function should be computable. We shall discover next week that this is not the case. Before we get to that we need to make very precise just what it means for a function to be computable. To do this we shall introduce the concept of recursiveness and device of the Turing machine.

7.1. Algorithms & Turing Machines

DEFINITION 148. An *algorithm* is simply a set of rules which govern the way we perform some calculation.

Less formally, an algorithm is just a set of instructions that we follow in order to perform a calculation. An algorithm is different from the function, the calculation for which it provides instructions. For example, consider the function of addition. There may be many different ways (i.e., algorithms) of calculating that function, but the function itself (i.e., addition) remains the same.

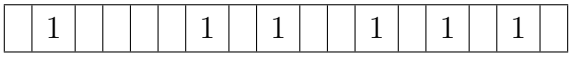
A Turing machine give us a *canonical way* of recording those instructions.

The set up is very simple.

- Image that you have an infinitely long piece of tape that is divided up into squares which we call *cells*.

- Each of the squares is either blank or has the numeral 1 printed in it.

We might represent this as follows:

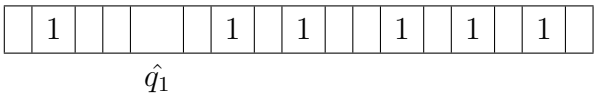


The Turing machine has something called a *head*. This is, in a sense, where the calculation is being done.

At any particular point in the calculation, the *head* is:

- sitting above a particular *cell*;
- in a particular *state*, q_1, which tells it what to do next.

We might indicate this as follows:[1]



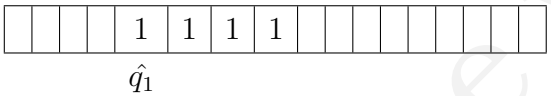
At a particular state it will be read what is on the cell below it and depending on what the cell has in it, the machine must:

- place a 1 in the cell below or make it *blank*; and then
- move either one cell to the *left* or *right*; or

A calculation will *terminate* or *halt* when the instructions do not have anything more for it to do.

An example will probably provide the easiest way to see how this might work.

EXAMPLE 149. Let us suppose that our tape is such that the only non-blank cells are a finite string of 1's, of say length 4 and that the head of the Turing machine sits under the leftmost 1.



This gives us the initial condition of the tape. Now let us define an algorithm which the Turing machine will perform. Let us say that we want to make the tape blank. What needs to happen?

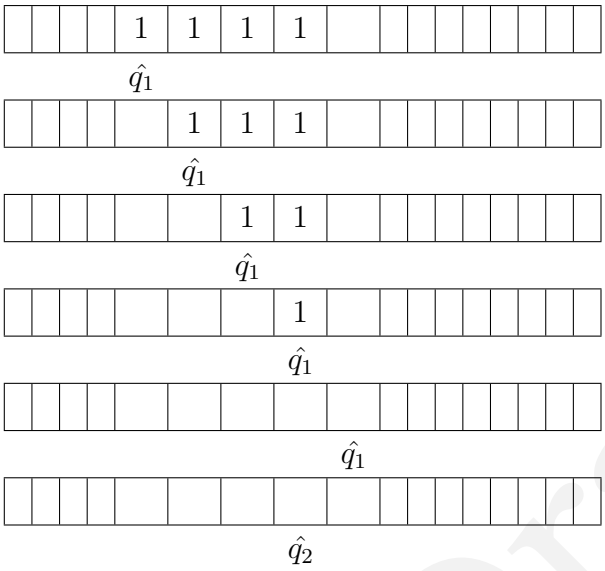
Essentially, we need to move along to the right blanking out all of the ones. After that we can stop. So let us propose the following instructions:

- In state q_1 if the cell has a 1 in it, then
  - change it to a *blank;*

[1]Note that the square above the q_0 is wider than the others for no reason other than typesetting.

  - move to the *right*; and
  - stay in state q_1.
- In state q_1 if the cell has a 0 in it, then
  - leave it as a *blank* (blanking a blank cell leaves a blank)
  - move one cell to the *left* (it doesn't really matter); and
  - change to state q_2.

We can step through the instruction as follows:



Now at the final line, observe that head is in stage q_2. However, we do not have any further instruction regarding what we should do in stage q_2, so the calculation halts at this point.

7.1.1. A Notation System. Observe that the instructions given above for this machine took up a lot of space to convey relatively little information. All we needed to know was what do in each stage given a certain condition of the tape. This can be summarised in a 5-tuple.

In state	if the cell is an	then change it to a	move to the	and go to stage
q	$1/-$	$1/-$	L/R	q'

In this way we could re-write the instructions above by simply writing:

- $q_1 1 - R q_1$
- $q_1 - - L q_2$.

We shall continue to adopt this system when describing Turing machines.

7.1.2. More Turing machines.

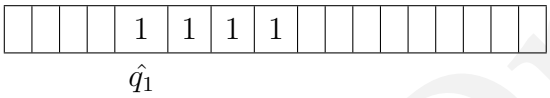
7.1.2.1. *Do Turing machines always halt?* An obvious question we might have is whether every Turing machine must come to a *halt*. This is not the case as we can easily illustrate. Let us take the machine above and make a seemingly trivial modification.

- $q_1 1 - Rq_1$
- $q_1 - -Lq_1$.

We stipulated that the rest of the tape was blank so the machine will simply keep working its way to the left and since the tape is infinite the tape it will never halt.

7.1.2.2. *Some simple arithmetic functions.* The function we considered doesn't do anything interesting. Let us now try to represent a couple of functions from arithmetic.

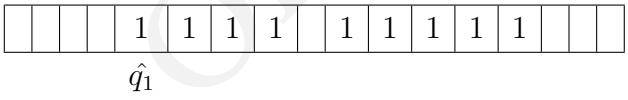
To do this we need some agreed way of representing natural numbers on the tape. Let us represent the number n by $n+1$ many consecutive 1's on the tape. And if n is the output of the function, let us represent this fact by having the head of the machine come to a halt over the leftmost 1. Thus if 3 were the output of some machine, then this would be represented as:



Now say that we wanted to define a function which added two numbers together. We then need to able to present more than one number to the Turing machine. To represent some n-tuple $\langle m_1, ..., m_n \rangle$, we shall place

- $m_1 + 1$ many 1s
- followed by a blank
- and then
- $m_2 + 1$ many 1s
- followed by a blank and
- ...
- finally $m_n + 1$ many 1s

along the tape. Thus $\langle 3, 4 \rangle$ would be represented as:



where the calculation begins in state q_0.

EXAMPLE 150. ($+1$) This function takes a number and adds 1 to it.

- $q_1 11 R q_1$
- $q_1 - 1 L q_2$
- $q_2 11 L q_2$
- $q_2 - - R q_3$

The idea is that we go to then end of the row, add a new one and then turn back.

EXAMPLE 151. $(m+n)$ This function takes two values and returns their sum.

- $q_1 11 R q_1$
- $q_1 - 1 R q_2$
- $q_2 11 R q_2 2$
- $q_2 - - L q_3$
- $q_3 1 - L q_4$
- $q_4 1 - L q_5$
- $q_5 11 L q_5$
- $q_5 - - R q_6$

The idea here is that we go through the tuple changing the blank between m and n into a one. Then we reach the right end, pull two 1's off it and return to the left.

Instead of writing $q_n 1 - q_m$, we shall now write: n 1 - m. Call this *streamlined notation*.

EXAMPLE 152. This function takes a number n and returns the tuple $\langle n, n \rangle$.

1 1 1 R 1

1 - - L 2

2 1 1 L 3

3 1 1 R 4

4 1 - R 5

5 - - R 6

6 - 1 L 7

7 - - L 7

7 1 1 L 8

8 1 1 R 9

9 1 - R 10

10 - - R 10

10 1 1 R 11

11 1 1 R 11

11 - 1 L 12

12 - - L 12

12 1 1 L 13

13 1 1 L 13

13 - - L 7

8 - - R 14

14 1 1 R 14

14 - 1 R 15

15 - 1 R 15

15 1 1 L 16

16 1 - R 17

17 1 1 R 17

17 - 1 L 19

19 1 1 L 19

19 - - L 20

20 1 1 L 20

20 - - R 25

3 - - R 25

The idea here is basically a lot of zig-zagging.

7.2. Gödel's schema

We now define an alternative way of describing algorithms which can be used to calculate functions. As we have seen the Turing machine system is extremely cumbersome for the purposes of describing often quite simple functions.

The next approach was developed by Gödel. It is much easier to define function using this technique, although the relationship with the intuitive notion of computability is probably less obvious.

It will turn out that each of these methods of describing algorithms is able to produce the same function.

This time we are going to build up algorithms using:

- basic functions; and
- rules for taking one function to another.

Any function that can be defined in this way is said to be a recursive function.

This should be familiar. This is another example of a definition by recursion, although the sense of recursion is slightly different in the case of the definition.

7.2.1. Basic functions.

7.2.1.1. *Notation.* We shall write $f : \omega \to \omega$ to mean that f is a function which from the natural numbers (ω) to the natural numbers. We shall write $f : \omega^n \to \omega$ to mean that f is a function which takes n-tuples of natural numbers and returns a natural number. We shall write $f : \omega \rightharpoonup \omega$ (and $f : \omega^n \rightharpoonup \omega$) to indicate that f is a partial function taking natural numbers (n-tuples of natural numbers) to natural numbers. It is partial in the sense that it may not be defined for every natural numbers (n-tuple of natural numbers); i.e., its domain may be a proper subset of ω ($dom(f) \subsetneq \omega$).

We shall call the functions defined using the schema below, *Gödel recursive functions*.

(1) The *zero* function, $z^m : \omega^m \to \omega$ is such that $z^m(\langle n_1, ..., n_m\rangle) = 0$ for all $n_1, ..., n_m, m \in \omega$.
(2) The *successor* function $s : \omega \to \omega$ is such that $s(n) = n + 1$.
(3) The *projection* function $\pi^m_k : \omega^m \to \omega$ is such that for $n_1, ..., n_m$ with $k \leq m$

$$\pi^m_k(\langle n_1, ..., n_m\rangle) = n_k.$$

None of these function is particularly exciting. They are just the basic cases.

REMARK. We shall also include a z^0 function which is the zero-place version of z^n. It takes no arguments (at all) and always returns the value 0.

7.2.2. Rules for making new functions. There are three processes for taking recursive function and constructing more complex recursive functions:

(1) Composition;
(2) Primitive recursion; and
(3) Minimisation.

7.2.2.1. *Composition.* Suppose we have Gödel recursive $f : \omega^m \rightharpoonup$ and $g_1, ..., g_m : \omega^k \rightharpoonup \omega$. Then there is a Gödel recursive function $h : \omega^k \rightharpoonup \omega$ such that

$$h(\langle n_1, ..., n_k\rangle) = f(\langle g_1(n_1, ..., n_k), ..., g_m(n_1, ..., n_k)\rangle).$$

REMARK 153. The idea of this function is that it allows us to compose function so that we may apply the result of a function the result of another function.

7.2.2.2. *Primitive recursion.* Suppose we have Gödel recursive $f : \omega^m \rightharpoonup \omega$ and $g : \omega^{m+2} \rightharpoonup \omega$. Then there is a Gödel recursive function $h : \omega^{m+1} \rightharpoonup \omega$ such that

$$\begin{aligned} h(\langle 0, n_1, ..., n_m\rangle) &= f(\langle n_1, ..., n_m\rangle) \\ h(\langle k+1, n_1, ..., n_m\rangle) &= g(\langle h(\langle k, n_1, ..., n_m\rangle), k, n_1, ..., n_m\rangle). \end{aligned}$$

This one arguably looks more complicated than it really is. An example might help.

EXAMPLE 154. Let's define addition. First we observe the following facts about addition:

$$\begin{aligned} 0 + n &= n \\ (k+1) + n &= (k+n) + 1. \end{aligned}$$

Both of these should be obvious. Moreover, we see that this is quite close to the form that we have above. The main thing we need is a function which takes a number an adds 1 to it. And we have one of these: the successor function. So we get something like this.

$$\begin{aligned} plus(\langle 0, n\rangle) &= \pi^1_1(\langle n\rangle) \\ plus(\langle k+1, n\rangle) &= s(\pi^3_1(\langle plus(k,n), k, n\rangle)). \end{aligned}$$

To emphasise that we are defining addition we write it as a function out the front of two arguments.

Observe that we have used both composition and primitive recursion to define this function.

7.2.2.3. *Minimisation.* Given a Gödel recursive function $f : \omega^{m+1} \rightharpoonup \omega$, there is a Gödel recursive $g : \omega^m \rightharpoonup \omega$ such that

$$\begin{aligned} g(\langle n_1, ..., n_m\rangle) &= \text{the least } k \text{ such that } f(\langle n_1, ..., n_m, k\rangle) = 0 \\ &\quad \text{and for all } l < k \ f(\langle n_1, ..., n_m, l\rangle) \text{ is defined.} \end{aligned}$$

Any function that can be described from the basic functions using these rules is a a *Gödel recursive* function.

REMARK 155. Note that this kind of minimisation is **not** the one which takes a tuple $\langle n_1, ..., n_k\rangle$ and returns the least element of that tuple.

7.2.3. Some examples of functions.

EXAMPLE 156. Let's define the function from Week 1 which took a number n and returned the triangular number with side n. I.e., we want the function $f : \omega \to \omega$ such that

$$f(n) = n + (n-1) + ... + 1.$$

(We shall take it that $f(0) = 0$.)

Now we can think of this function as being calculated in stages as follows:

$$\begin{aligned} f(0) &= 0 \\ f(n+1) &= (n+1) + f(n). \end{aligned}$$

This has the right kind of shape for primitive recursion. From here, we can then represent it in the proper notation as follows:

$$\begin{aligned} tria(0) &= z^0 \\ tria(n+1) &= f(\langle tria(n), n\rangle) \end{aligned}$$

where

$$f(n,m) = plus(\langle n, s(m)\rangle).$$

Note that in defining this function, we have used the function $plus$ which we have already defined.

EXAMPLE 157. Let us define a function $pre : \omega \to \omega$ which takes a number $n+1$ and returns its predecessor n if there is one and 0 otherwise. Then we want something like this:

$$\begin{aligned} pre(0) &= z^0 \\ pre(n+1) &= g(\langle pre(n), n\rangle) \end{aligned}$$

where $g(m,n) = \pi^2_2(\langle m, n\rangle) = n$.

EXAMPLE 158. Let us define a function $sub : \omega^2 \to \omega$ which takes n and m and returns $m - n$ if it is defined and 0 otherwise. We define this as follows:

$$\begin{aligned} sub(\langle 0, m\rangle) &= \pi^1_1(\langle m\rangle)\ (= m) \\ sub(\langle k+1, m\rangle) &= g(\langle sub(\langle k, m\rangle), k, m\rangle) \end{aligned}$$

where $g(\langle v, u, w\rangle) = pre(Id^3_1(\langle v, u, w\rangle) = pre(v)$.

EXAMPLE 159. Let's define the function $squ : \omega \to \omega$ which takes a number and squares it. Getting started, we note that

$$0^2 = 0$$

and that

$$(n+1)^2 = n^2 + 2n + 1.$$

EXAMPLE 160. So we can put this into the notation as follows:

$$\begin{aligned} squ(0) &= z^0 \\ squ(n+1) &= f(\langle squ(n), n\rangle) \end{aligned}$$

where $f : n^2 \to n$ is such that

$$f(\langle n, m\rangle) = plus(\langle n, plus(\langle plus(\langle m, m\rangle), 1\rangle)\rangle)$$

We can also represent relations using functions defined by algorithms.

EXAMPLE 161. We define a function $gr0 : \omega \to \omega$ which takes n and returns the value 1 if n is greater than 0 and 0 otherwise. We define this as follows:

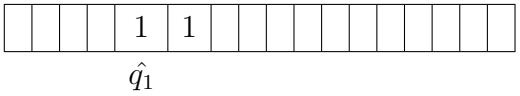
$$gr0(n) = sub(\langle pre(n), n\rangle).$$

EXAMPLE 162. Let $gre(m, n)$ be the function which returns 1 if n is greater than m and 0 otherwise. We use the sub function and composition to get:

$$gre(\langle m, n\rangle) \quad = \quad gr0(sub(\langle m, n\rangle).$$

7.3. Exercises.

EXERCISE 163. Step through the first 26 steps of the calculation of the function in Example 152 given starting condition.



EXERCISE 164. Write a Turing machine which doubles a number n. Use the notation system of Example 152. Do not use state 0.

EXERCISE 165. (*Harder* - well ... tedious) Write a Turing machine which takes two numbers m and n and returns their product.

EXERCISE 166. Using the Gödel notation, describe an algorithm representing a function $times : \omega^2 \to \omega$ which take m and n and returns $m \times n$.

EXERCISE 167. Using the result from Week 1, define an alternative algorithm which gives triangular numbers. (You may use other functions which we have already defined.)

EXERCISE 168. Using the Gödel notation, describe an algorithm which represents a function $exp : \omega^2 \to \omega$ which takes m and n and returns m^n.

EXERCISE 169. Using the Gödel notation, describe an algorithm which represents a function $fac : \omega \to \omega$ which takes a number and returns $n!$ (i.e., $n \times (n-1) \times ... \times 2 \times 1$).

EXERCISE 170. Using Gödel notation, define a function $geq : \omega^2 \to \omega$ which takes m and n as arguments and returns 1 if n is greater than m and 0 otherwise.

EXERCISE 171. Using Gödel notation, define a function $equ : \omega^2 \to \omega$ which takes m and n and returns 1 if they are equal and 0 otherwise.

CHAPTER 8

Recursion theory 2

8.1. Equivalence of Turing machines with Gödel schema

Last week, we looked at two ways of *formalising* our intuitive idea of computation. On the one hand, we looked at Turing machines which were based on extremely simple rules; and on the other, we looked at the Gödel schema system which provided an elegant means of representing common functions from the theory of arithmetic.

To start this week off, we are going to provide an outline of the proof which shows that these two systems are, in some salient sense, equivalent. This is interesting since it shows us that two, seemingly quite different, means of formalising computation actually end up doing the same thing.

We only provide a sketch because the full details of the proof are quite tedious and lengthy.

The first thing we need is a way of comparing the two systems. At face value, this isn't going to be possible. Turing machines talk about lengths of tape, while the Gödel schema are concerned with functions on the natural numbers.

8.1.1. Numbers for Turing machines. Last week, we saw a way of representing numbers and tuples using a tape representation. This gets us on our way to representing Gödel functions using Turing machines: we have, so to speak, the matter for our calculation.

In addition to this, we need some way of coding up the functions we can represent using Gödel schema. Suppose we had some function $f : \omega^n \to \omega$ defined using the Gödel schema. Our goal is to find a means of representing this situation as the starting condition on the tape.

You have probably noticed that constructing Turing machines relies a lot on counting sequences of blank squares. For example, when we get to the end

of a tuple representation we can tell that this has occured because we have more than one consecutive blank square.

With this is mind, we might make a few modications to our representations of tuples an numbers so that we can get a clearer representation of the syntax of the Gödel schema.

So first of all, we are going to need some way of representing the primitive symbols of the Gödel schema. Thus we need ways of representing:

- z^m;
- s; and
- π^n_k.

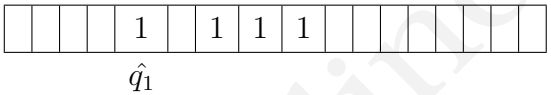
Moreover, we also need to represent the operations which allow us to build more complicated functions using:

- composition;
- primitive recursion; and
- minimisation.

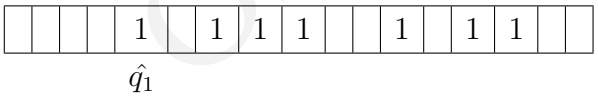
Intuitively speaking, this kind of stuff is obviously different from numbers and tuples. So it would be convenient if we had a simple way of discerning the two.

8.1.1.1. *A different way of representing numbers and tuples.* So our goal here is to make a coding that clearly distinguishes numbers and tuples from symbols. We are going to make use of multiple blanks for this purpose.

Let us represent a *number* n on the tape by writing a 1 followed by a blank and then $n+1$ many 1's after that. So the following diagram represents the number 2.



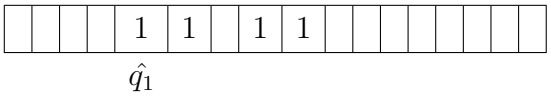
Let us represent a tuple of length m by placing a sequence of m number representations each separated by 2 blank cells. Thus the following situation represents the tuple $\langle 2, 1\rangle$.



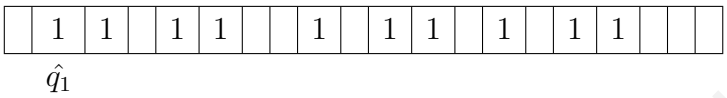
So this makes things a little more complicated for representing numbers, but in the end it will be easier to represent the Gödel schema.

8.1.1.2. *A way of representing Gödel schema.* Now for every m we know that we have a function $z^m : \omega^m \to \omega$ which takes a tuple of length m and always returns the value 0.

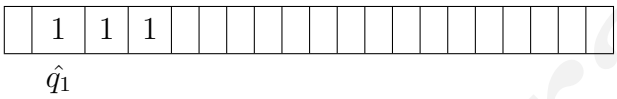
We shall represent this by 2 cells containing 1 followed by a blank followed by n-many 1's. Thus we would represent the function z^2 by the following:



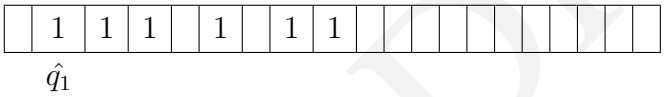
Now we really want to represent the situation when this function is applied to some tuple. To do this we shall place the code of the function to the left of the representation of the tuple with two blank cells between them. Thus we migh represent $z^2(\langle 1, 1\rangle)$ as follows:



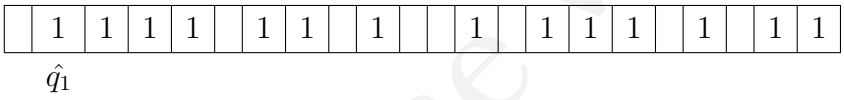
In a similar fashion, we represent the function $s : \omega \to \omega$ by placing three 1's consecutively on the tape. Since the s function only deals with arity 1, there is no need to deal for further information. Thus we represent it as follows:



And we can represent that function applied to the number 1 as follows:



Next we represent the projection function $\pi^n_k : \omega^n \to \omega$. We do this by placing 4 1's on the tape followed by a blank followed by n $1's$ followed by a blank followed by k 1's. Thus we would represent $\pi^2_k(\langle 2, 1\rangle)$ as follows:



So we now have a way of represting all of the basic function from the Gödel schema. We now describe how to represent the processes for constructing more complex recursive functions.

We start with composition. So suppose we have $f : \omega^m \rightharpoonup \omega$ and $g_1, ..., g_m : \omega^k \rightharpoonup \omega$. Then by the composition rule, we know there is a recursive function h such that:

$$h(\langle n_1, ..., n_k\rangle) = f(\langle g_m(\langle n_1, ..., n_k\rangle), ..., g_m(\langle n_1, ..., n_k\rangle)\rangle).$$

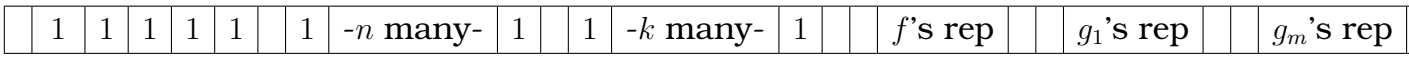
In order to represent this, it should be clear that all we need is to:

(1) represent the arity m;
(2) represent the arity k;
(3) represent the function f; and
(4) represent each of the functions $g_1, ..., g_m$.

So let us represent composition by placing 5 1's, then a blank followed by:

(1) m $1's$ followed by a blank; then
(2) k $1's$ followed by $\underline{2}$ blank cells; then
(3) the representation of the function f followed by 2 blanks; then
(4) for each $1 \leq i \leq m$, the representation of g_i each separated by 2 blank cells.

Thus, in general a composition will look like:



$\hat{q}_1$

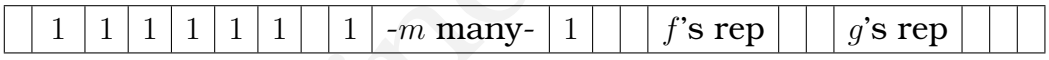
Now we represent primitive recursion in a similar fashion. Suppose we have $f : \omega^m \rightharpoonup \omega$ and $g : \omega^{n+2} \rightharpoonup \omega$. Then there is a recursive function $h : \omega^{m+1} \rightharpoonup \omega$ such that:

$$\begin{aligned} h(\langle 0, n_1, ..., n_m \rangle) &= f(\langle n_1, ..., n_m \rangle) \\ h(\langle k+1, n_1, ..., n_m \rangle) &= g(\langle h(\langle k, n_1, ..., n_m \rangle), k, n_1, ..., n_m \rangle). \end{aligned}$$

We represent such an h by placing 6 $1's$ on the tape followed by a blank followed by:

- m many 1's followed by 2 blanks; then
- the representation of the function f; then
- the representation of the function g.

Thus in general, the tape will look like the following diagram.



$\hat{q}_1$

And finally we come to minimisation. Suppose we have a recursive function $f : \omega^{m+1} \rightharpoonup \omega$. Then we know that there is a recursive function

$$g(\langle n_1, ..., n_m \rangle) = \text{the least } k \text{ such that } f(\langle n_1, ..., n_m, k \rangle) = 0.$$

We represent such a g by placing 7 $1's$ on the tape followed by a blank and then:

- m 1's followed by 2 blanks; then

- the representation of g.

Thus, in general a representation of the minimisation function will look like this:

	1	1	1	1	1	1	1		1	-m many-	1			g's rep			
	$\hat{q}_1$																

So we now have a way of representing any function that can be defined using the Gödel schema and the tuple that is fed into such a function. Unfortunately, this is just the beginning of the job.

8.1.1.3. *Completing the proof.* With this representation, we have the scene set for completing the proof. As you can see, just setting things up takes up a reasonable amount of space. So we'll satisfy ourselves here, by sketching what needs to be done.

Now what we want to show here is that a Turing machine can calculate any function that can be calculated using the Gödel schema. We now have a means of representing any function constructed using the Gödel schema and in order to present an argument tuple to it, we simply place that tuple two blank cells after the representation of the function.

In order to complete the proof we need to construct a Turing machine which takes:

- the representation of a Gödel schema algorithm; and
- a tuple of numbers,

and then calculates what that Gödel schema algorithm would have done.

So to do this we "simply" need to build a function that will:

- *recognise* which Gödel schema rules or basic function is being employed; and then
- *run* that function.

The *recognition* part is easy. We simply construct a Turing machine which recognises how many 1's are at the beginning of the code as this is how we tell which part of the Gödel schema has been used. For example, we might start as follows:

1 1 1 R 2

2 1 1 R 3

3 1 1 R 4

4 1 1 R 5

5 1 1 R 6

6 1 1 R 7

7 1 1 R 8

8 1 1 R halt ; So we halt if there are more than 7 1's as we don't allow for this in the code.

2 1 1 R halt ; We also halt if there is only 1 1, since then we don't have a function code.

3 - - R 31 ; This was the code (i.e. 2 1's) for the z^m function, so we now run that code.

...

4 - - R 41 ; This was the code (i.e., 3 1's) for the s function, so we now run that code.

...

...

8 - - R 81 ; This was the code for the minimisation function, so now we run that code.

We now describe how to *run* the successor function (basically since it's the easiest). So suppose we have just gone to state 4 having passed 31's. We are now in state 41. We then proceed as follows:

41 - - R 42 ; There should be two blanks in between the function and its argument.

42 1 1 R 42 ; We go right until we reach the end of the row of 1's.

42 - 1 L 43 ; We then place a 1 in the blank cell and move to the left.

43 1 1 L 43

43 - - R halt ; We reach the end of the 1's and halt under the leftmost one of them.

The z^m function is similarly easy to describe.

For the rest of the functions we shall content ourselves with a high-level description of how to construct the machine:

- Projection (π^n_k): We read across to the k-sequence of 1's and then go back and forth erasing an element of k and an element of the tuple until we reach the k^{th} element. We then use the n-sequence to erase

the rest of the tuple. Finally we zig-zag the k^{th} element of the tuple back to the starting point.

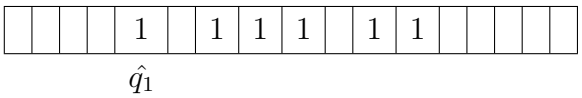
- Composition: We may need some space for this calculation. Thus for each of the g_i where $1 \leq i \leq m$, we take g_i's code and a copy of the tuple $\bar{n}$ and place them to the right of the initial setup. We then run g_i on $\bar{n}$. We then leave the output to the right and repeat for each of the inputs until we have a new tuple. We move this tuple into position so that we can apply f to it; and we do this.
- Primitive recursion: This is definitely the ugly one. In some sense, it's just a more general version of the multiplication algorithm you have already constructed. Remember that primitive recursion is designed, loosely speaking, so that we can perform k many repetitions of a certain calculation. Thus we shall keep on, so to speak, unfolding the calculation to the right which will (eventually) reduce the k parameter to 0, at which point we may apply the f function and get an output. We then work backwards to the left, putting the input back through the g function until we get to the beginning again and that gives us our output.
- Minimisation: This time we just keep running the function f on successively greater inputs. A tally will need to be kept somewhere to the right on the tape. At each stage we check whether the output is 0. If it is, then the tally is our output and we rearrange the tape accordingly. If we never reach such a point, then the calculation will not halt.

Now the full proof of this would take up a great deal more space. Moreover, I don't want to trivialise that activity, but (hopefully) you should be able to see that the task can be completed. You should know enough about Turing machines to see that this would be a challenge but one that can be completed. I won't set this as an exercise, although it's certainly a good thing to try to do at some point.

8.1.2. Tapes for Gödel schema. Now we want to show the converse of the above. We want to show that anything that can be done using a Turing machine can also be done with the Gödel schema. Again, we'll only lay the groundwork for the proof here.

8.1.2.1. *A way of representing Turing machines.* As before, our initial problem is finding a way of comparing the two approaches. This time we need some way of representing the *tape* of the Turing machine using the natural numbers. There is a very simple way of doing this.

Suppose we hade a tape and head in the following position:



We could represent this situation by the number:

$$10111011$$

Thus, we take the head and then work our way t the right using a 1 to represent a 1 on the tape and using a 0 to represent a blank.[1] However, it will actually be convenient to reverse the direction of the representation. Thus we shall code the tape above as:

$$11011101$$

So that gives us a way of representing the tape. We now need to represent the *instructions* for the Turing machine. The technique we use is known as *Gödel coding*. We shall be using this technique again next week and the week after that.

The essential idea we rely on is the following basic fact from number theory:

FACT 172. *Every natural number n is such that it has a unique prime decomposition; i.e., there is a unique set of prime numbers $p_1, ..., p_m$ and $k_1, ..., k_n \geq 1$ such that:*

$$n = p_1^{k_1} \times ... \times p_m^{k_m}.$$

A couple of examples might make this clearer. Consider the number 63. It should be easy to see that:

$$\begin{aligned} 63 &= 9 \times 7 \\ &\quad 3^2 \times 7. \end{aligned}$$

[1] Note that we can only represent finite sequences using this technique. To represent infinite sequences, we'd need the real numbers and we have not designed our Gödel schema to deal with them.

Consider the number 48. Clearly, we have

$$\begin{aligned} 48 &= 6 \times 8 \\ &= 3 \times 2 \times 2^3 \\ &= 3 \times 2^4 \end{aligned}$$

We shall represent a set of instructions for a Turing machine in two stages:

(1) we represent each of the lines; then
(2) we put all the lines together.

A typical *line* of Turing machine instructions will be of the form:

n ? ? D m

where n and m are natural numbers; $?$ is either 1 or $-$; and D is a direction (either L or R). First we need to represent all the parts as natural numbers. Let us represent blank by 2; L by 1 and R by 2. Then a line like:

34 1 - R 45

would be represented as follows:

34 1 2 2 45.

Using Gödel coding we may represent this as follows:

$2^{34} \times 3^1 \times 5^2 \times 7^{45}$.

Essentially we go through use the first four prime numbers to code up the line. It will give us a very big number, but, most importantly, it's a number from which we can recover the instructions.

To represent a *sequence* of lines of code, we simpy repeat the trick. Suppose we have some lines of instructions such that their code is given in the following table:

Instruction	Code number
46 1 2 2 78	c_1
67 1 1 1 98	c_2
...	
98 1 2 1 23	c_m

We then represent this sequence $(c_1, ..., c_m)$ of code numbers by:

$$2^{c_1} \times 3^{c_2} \times ... \times p_m^{c_m}$$

where p_m is the m^{th}prime number.

Thus we now have a way of representing the algorithm (instructions) for a Turing machine as well as its tape.

8.1.2.2. *Completing the proof.* Now this time, we want to define a function using the Gödel schema which takes a code for a tape and a code number for a Turing machine and returns the output of that machine for that input tape.

Once again, I'm only going to describe this at a *very* high level. The full details are worth working but are beyond the scope of this module.

Essentially, we want a function which:

(1) Decodes the instruction set and finds out what to do at state 1.
(2) Performs the calculation at that state and records a number for the new position of the head on the tape and the new state of the head.
(3) It should then repeat the calculation for this new state and position until a halting position is reached. This will involve *minimisation* and we'll want to use the tape code as k here.

Of course, this is extremely superficial. If you are interested in completing the exercise, you are probably best off describing the process completing the Turing calculation using simple "if ..., then —; otherwise _ _ _" instructions and then figuring out how to get Gödel schema to do this.

8.2. The Church-Turing thesis

So now we have seen two different approaches which formalise our intuitive (or informal notion) of (effective) computation or calculation. Moreover, we have seen (at a high level) that these two approaches are actually equivalent.

There have also been other attempts to formalise effective computability. These include:

- Church's Lambda calculus;
- Kleene's equational representation;
- register machines;
- Markov algorithms; and
- combinatorial logics.

We won't be looking at any of these other systems, although they are easy enough to find. The interesting thing is that every one of these systems can be show to be equivalent to every other.

So every serious attempt to formalise the notion of effective computability has ended up with the same class of functions. These are known as the *recursive* functions. We've used this term before, but now we'll make our usage more precise.

DEFINITION 173. Let us say that function is *recursive* if it has an algorithm in any one (and thus all) of the canonical formalisations.

Thus if we can make a Turing machine describing a particular function, then that function is recursive.

But we're going to take a step further than this now. Given that every approach to formalising effective computability has come to the same thing, we are going to propose the following thesis:

THESIS: (Church/Turing) The set of recursive functions *is* the set of effectively computable functions.

Observe that this is not a theorem, it is an *empirical thesis*. We have not proven that it is correct from principles which are incontrovertible. Rather we have come to think that it is true on the basis that its assmption is (very convenient) and that there are no known counterexamples.

Given that effective computability is an informal notion, we perhaps shouldn't be too surprised or disappointed by this. Effective computability is a very worldly concept and thus any attempt to approximate or describe it is limited by how much we can know about such things. Arguably, we are in a similar situation in physics when we attempt to describe the laws which govern the physical world.

It is also very useful. It allows us to move between the informal notion of a function or effective calculation which we can peform using simple rules to the formal notion of recursiveness. We shall make use of this in future weeks.

To take an example, however, consider the strings of symbols which we can form in a finite language and consider how we verify that a such a string is well-formed or not.

Can you think of a pen-and-paper algorithm which would tell you that a string is well-formed? Hopefully, the answer is yes, but it's worth thinking about in detail. In this case, we may use Church's thesis to assert that such a function is actually recursive. The more detailed verification of this

fact would involve finding a suitable coding for the strings and then showing that there is a Gödel schema or Turing machine algorithm which can peform such a calculation.

8.3. Limitations in the theory of recursion

8.3.1. Are all functions from ω to ω recursive? An obvious question we might want to know is whether or not all of the functions taking arguments in ω and returning values in ω are recursive. The answer to this question is negative.

To see this, we observe that there are countably many recursive functions. Obviously, there are countably many of them. To see that there are at most countably many recursive functions, observe that any set of instructions (i.e., an algorithm) can be represented (as we saw above) by a natural number. Since every recursive function is determined by at least one algorithm, there are at most countably many recursive functions.

On the other hand, let us consider how many functions there are taking arguments from ω and returning values from ω; i.e., what is the cardinality of the functions $f : \omega \to \omega$.

Now any of the functions $f : \omega \to \omega$ can be used to represent a unique subset $F \subseteq \omega$ as follows. We let

$$n \in F \ \leftrightarrow f(n) = 0.$$

Thus if $f(7) = 0$, then $7 \in F$; but if $f(7) = 89$, then $7 \notin F$. But we know from Corollary 136 that the collection of all subsets of natural numbers is not countable. Thus, the collection of all functions from ω to ω is also uncountable.

8.3.2. Recursive and partial recursive functions. Recall the distinction between total and partial functions. We shall say that a function $f : \omega^2 \to \omega$ enumerates a (countable) set of functions $G = \{g_n \mid n \in \omega\}$ if for every $g_n \in G$ there is some $n \in \omega$ such that

$$g_n(m) = f(n, m)$$

for all $m \in \omega$. In other words, for any imput m, g_n gives the same output as $f(n, \dots)$: they compute the same function.

It would be convenient to have some way of enumerating the total recursive functions. Thus we might ask whether there is a total recursive function which enumerates all of the total recursive functions? Perhaps surprisingly, the answer is no.

THEOREM 174. *There is no total recursive function which enumerates all of the total recursive functions.*

PROOF. Suppose for *reductio* that $f : \omega^2 \to \omega$ is a total recursive function which enumerates all the total recursive functions. Let $g : \omega \to \omega$ be such that for all m,

$$g(m) = f(m, m) + 1.$$

Since f is total recursve, g is clearly recursive too. We just used successor and composition.

We claim that g cannot be in the enumeration. Suppose it were. Then there would be some k such that for all $m \in \omega$

$$g(m) = f(k, m).$$

But then if we consider what g does with value k, we get the following:

$$f(k, k) = g(k) = f(k, k) + 1$$

which is a contradiction. Thus g is not in the enumeration and thus there is no total recursive function enumerating the total recursive functions. □

REMARK. Observe that the argument used above is basically the same as the one used in proving Cantor's theorem. We adjusted the values of the enumeration function down the diagonal to get a function that could not have been on the list. This technique is ubiquitous in mathematical logical. It is often known as *diagonalising out*.

8.3.2.1. *Partial recursive functions.* There is a kind of way around this problem. Rather than considering the total recursive functions, we might consider the functions which are partial and recursive.

For example, in Section 7.1.2.1, we saw an example of a Turing machine that did not halt. Moreover, we saw that it was easy to think of examples of minimisation that never halted since they never came to a value for k which made the overall function output 0.

We thus introduce a more embrasive category of *partial recursive functions* which include these non-halting cases and also the total recursive functions.

We can give an enumeration the *partial recursive functions*. For example, might use the code number of the Gödel schema algorithm of a function. Suppose that n is the code of some Gödel schema. We shall denote the function code by n as φ_n.

To take an example, suppose that the code of the *multiplication function* is e for some $e \in \omega$. Then we have,

$$\varphi_e(m, n) = m \times n$$

for $m, n \in \omega$.

Given that partial recursive functions are not always defined it will be helpful to introduce a notion of equality between them. The following is useful.

DEFINITION 175. Let φ_s and φ_t be partial recursive functions. We say that $\varphi_s \simeq \varphi_t$ if

- for all m for which φ_s is defined, $\varphi_s(m) = \varphi_t(m)$; and
- for all m for which φ_t is defined, $\varphi_s(m) = \varphi_t(m)$.

So the basic idea here is that we identify two partial recursive function if they produce outputs for exactly the same domain of natural numbers and that the produce the same outputs over that domain.

Now we might ask the question: is there a partial recursive function which enumerates all of the partial recursive functions? In this case the answer is yes.

Appealing to Church's thesis, we argue that we could construct a pen-and-paper algorithm which takes the code number e of some algorithm as an argument and then behaves like φ_e for any imput m. More formally, a partial recursive function φ_k (for some k) such that for all φ_e (i.e. partial recursive functions):

$$\varphi_k(e, \dots) \simeq \varphi_e.$$

Or in other words for all m for which φ_e is defined $\varphi_k(e, m) = \varphi_e(m)$ and for other m, $\varphi_k(e, m)$ is not defined. More formally we might write:

$$\varphi_k(e,m) \quad = \quad \begin{cases} \varphi_e(m), & \text{if } \varphi_e(m) \text{ halts.} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Such a function is known as *universal* (or in the specific case of a Turing machine, a *universal Turing machine*).

We should wonder why the problem for total recursive function does not recur here. Let's step through the argument to see why.

PROOF. (ABORTIVE) As before let us suppose for "*reductio*" that there is a partial recursive function φ_k which enumerates the partial recursive functions. Then let us consider the partial recursive function φ_j which is defined such that for all m :

$$\varphi_j(m) = \begin{cases} \varphi_k(m,m)+1, & \text{if } \varphi_k(m,m) \text{ halts.} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

It should be obvious that φ_j is also a partial recursive function.

Thus φ_j has a code number; i.e., j. Let us consider what happens when φ_j is applied to j (its own code number),

$$\varphi_j(j) = \begin{cases} \varphi_k(j,j)+1, & \text{if } \varphi_k(j,j) \text{ halts.} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Supposing that $\varphi_k(j,j)$ halts we would have

$$\varphi_k(j,j) = \varphi_j(j) = \varphi_k(j,j)+1.$$

But this is obviously a contradiction, so it cannot be the case that $\varphi_k(j,j)$ halts. □

8.3.3. The halting problem. We now consider the question: is there a recursive function which can take the code of a partial recursive function and tell us whether or not the function always halts?

Clearly this would be a handy thing to have. Given some set of instructions (suitably coded) such a function could tell whether or not some programme would eventually halt. Intuitively speaking, when a computer crashes (or hangs) this is often because the underlying programme does not halt. It

would be nice to have a special programme that could look at other programmes and tell you whether or not they hang. This is known as the *halting problem*.

Let us call a function $f : \omega^2 \to \omega$ a *halt-checker* if it takes the codes of partial recursive functions and tells us whether or not for some input m that partial recursive function halts on m. More formally,

$$f(e,n) = \begin{cases} 1 & \text{if } \varphi_e(n) \text{ halts.} \\ 0 & \text{otherwise.} \end{cases}$$

Unfortunately this cannot be done.

We shall, in fact, show something a little stronger than this. Let us call a function $f : \omega \to \omega$ an *auto-halt-checker* if f is such that:

$$f(e) = \begin{cases} 1 & \text{if } \varphi_e(e) \text{ halts.} \\ 0 & \text{otherwise.} \end{cases}$$

We shall show that there is no (total) recursive auto-halt-checker from which it clearly follows that there is no halt checker.

THEOREM 176. *There is no auto-halt checker.*

PROOF. Suppose not. Then let $f : \omega \to \omega$ be a (total) recursive auto-halt checker; i.e., f is such that

$$f(e) = \begin{cases} 1 & \text{if } \varphi_e(e) \text{ halts.} \\ 0 & \text{if } \varphi_e(e) \text{ does not halt.} \end{cases}$$

Now let us define a partial function $\varphi_k : \omega \rightharpoonup \omega$ which is such that

$$\varphi_k(e) = \begin{cases} \text{undefined}, & \text{if } f(e) = 1. \\ 0 & \text{if } f(e) = 0. \end{cases}$$

To see that such partial recursive function exists, we appeal to Church's thesis by describing the algorithm required. Since f is total recursive, when we peform that calculation it will either terminate on 1 or 0. If it outputs 1, we give the machine instructions to loop. If it ouputs 0, we make that the ouput.

Let us see what happens when we present k (the code for φ_k) to the function φ_k.

Let us first suppose that $\varphi_k(k)$ is defined. Then by its definition, $\varphi_k(k) = 0$. Thus $f(k) = 0$. But this means that $\varphi_k(k)$ does not halt (by definition of f); and thus $\varphi_k(k)$ is not defined. This contradicts our assumption. Thus $\varphi_k(k)$ is undefined.

But this means (by definition of φ_k) that $f(k) = 1$; and thus by definition of φ_k, we see that $\varphi_k(k)$ halts and is thus defined. This contradicts our assumption that such a φ_k exists, so there is no such φ_k.

But there was nothing wrong with our definition of the partial recursive φ_k on the assumption that f is a total recursive function. So our assumption is wrong; and thus, there is no such total recursive f. □

COROLLARY 177. *There is no halt-checker.*

8.4. Recursive and recursively enumerable sets

In this section, we relate the concepts of recursion theory to sets.

DEFINITION 178. A set $A \subseteq \omega$ is *recursively enumerable* if A is the domain of a partial recursive function.

So if $n \in A$, there is (by Church's thesis) an effectively computable function which will be verify this fact.

Suppose A is recursively enumerable and that $n \in A$. We verify this as follows. Given that A is recursively enumerable, there is a partial recursive function, say φ_e such that A is the domain of φ_e. Or in other words, for all $n \in \omega$,

$$n \in A \;\Leftrightarrow\; \varphi_e(n) \text{ halts.}$$

DEFINITION 179. We say that $A \subseteq \omega$ is *recursive* if:

- A is recursively enumerable; and
- $\omega \backslash A$ is recursively enumerable.

This means that for any n we can verify whether **or not** it is in A. Given A is recursive, there are partial recursive functions φ_e and φ_f which have domains A and $\omega \backslash A$ respectively. To check whether $n \in A$, we simply run

both $\varphi_e(n)$ and $\varphi_f(n)$ simultaneously (or perhaps switching back and forth). Clearly, at most one of them can halt since we cannot have $n \in A$ and $n \in \omega \backslash A$ (i.e., $n \notin A$). Moreover, one of the functions must halt, since either $n \in A$ or $n \notin A$.

Naturally enough, this might lead us to ask the question: doesn't the same apply to recursively enumerable sets? The answer here is no. To see this it suffices to show the following:

THEOREM 180. *There is a $K \subseteq \omega$ which is recursively enumberable but not recursive.*

PROOF. Let $K = \{e \in \omega \mid \varphi_e(e) \text{ halts.}\}$. We claim that K is recursively enumberable but not recursive.

To see that K is recursively enumberable, we need partial recursive function $\psi(e)$ such that

$$\psi(e) \text{ halts. } \Leftrightarrow \varphi_e(e) \text{ halts.}$$

For this, we simply use the *universal* machine φ_k such that $\varphi_k(e, \dots) \simeq \varphi_e$. We let $\psi(e) \simeq \varphi_k(e, e) \simeq \varphi(e)$. Then if $\varphi_e(e)$ halts, then $\psi(e)$ halts (and gives the same output) and *vice vera*.

To see that K is not recursive, we need to show that $\omega \backslash K$ is not recursively enumerable. Suppose it was. Then there is a partial recursive function $\varphi_j(m)$ such that:

$$\varphi_j(m) \text{ halts } \Leftrightarrow \varphi_m(m) \text{ does not halt } \Leftrightarrow m \in \omega \backslash K$$

Now consider what happens if we input j into φ_j. We have:

$$\varphi_j(j) \text{ halts } \Leftrightarrow j \in \omega \backslash K \Leftrightarrow \varphi_j(j) \text{ does not halt.}$$

The first $\Leftrightarrow$ follows from the definition of φ_j; the second $\Leftrightarrow$ follows from the definition of K. But this is a contradiction, so there is no such partial recursive function. □

REMARK 181. Note that this proof is very similar to that of Theorem 176.

8.5. Exercises.

EXERCISE 182. Does every algorithm determine a partial recursive function?

EXERCISE 183. Can a partial recursive function be represented by more than one code number? Why/why not?

EXERCISE 184. Give a direct Cantorian proof that there are uncountably many functions $f : \omega \to \omega$; i.e., use a diagonal argument.

EXERCISE 185. Prove Corollary 177.

EXERCISE 186. Show that set $A \subseteq \omega$ is recursively enumerable iff it is the range of a total recursive function or the empty set. [Use Church's thesis to define algorithms, in an informal fashion, that will meet the requirements of the definitions. Note that we only need perform a finite part of a calculation at a time.]

EXERCISE 187. Take $A \subseteq \omega$. The *characteristic function* $\chi_A : \omega \to \omega$ *of* A is such that:

$$n \in A \;\Leftrightarrow\; \chi_A(n) = 0.$$

Show that a set $A \subseteq \omega$ is recursive iff ifs characteristic function is total recursive.

EXERCISE 188. Is there a partial recursive function which takes (codes of) sentences φ and returns the value 0 if $\models \varphi$? How would you show this? [Please feel free to use Church's thesis.] What does this say about the set of valid sentences? What about the set of sentences which are merely satisfiable, i.e., $\not\models \neg\varphi$.

CHAPTER 9

Arithmetic

9.1. Theories and axioms

We may describe a theory in first order logic by providing a set of *axioms* to *generate* it.

For example, we can construct theories for:

- arithmetic;
- set theory; and
- syntax.

The axioms are simply sentences of first order logic which:

(1) state things which are obviously true; and
(2) are sufficient to capture everything about the subject of our theorising.

So we would want our theory of arithmetic to have axioms that are obviously true and which capture everything about the subject matter: arithmetic.

The notion of *capturing* here is that of proving. We use the axioms as premises in, say a natural deduction proof. In our informal proof about triangular numbers in Week 1 we completed our proof on the basis of an induction axiom and some simple facts about number theory. A good theory would have been able to capture those simple facts and everything else.

We are going to learn in the next couple of weeks that the goal (2.) is actually impossible to achieve in most interesting cases. This phenomena is known as *incompleteness*. There are sentences which we can neither prove nor refute.

9.2. A sketch of the incompleteness theorem

Before we get into the full detail of this proof, which will take a while, I'll give you a quick and simplified sketch of the proof. This will give you an idea of

the phenomena involved and also allow us to pick out the important parts of the strategy involved in proving it.

So my goal is to show that there is a sentence γ which I can neither prove from my axiom system Γ nor refute in it.

But first consider two properties which seem quite clearly desirable for an axiom system:

- **Consistency:** we don't want to be able to prove $\bot$ from Γ. Otherwise, as we know, we could prove anything we liked and the our theory would not be very interesting.
- **Soundness:** if we can prove a sentence φ, the φ is true. Here we are just saying that if we can prove it then it better not be false. Clearly a theory of some subject ought to do this.

Now the axiom system Γ that we are going to be concerned with is a theory of arithmetic PE. This theory is quite strong. As we've seen above we can code up Turing machines using numbers. We can use a similar trick to code up sentences of arithmetic. Moreover, we shall see today that PE is actually strong enough to represent recursive functions and as we shall see a predicate $B(x)$ which says, loosely speaking, that x is the code of a provable sentence. Showing these things is where we need to do most of the work.

However, once these things are established, we can then go on to show that there is a sentence γ which says of itself (in some sense) that it is not provable. (This is a little like the famous liar sentence: this sentence is not true.) With this in hand we can complete the argument.

9.2.1. An informal version of the agument. We show, informally here that there is a sentence which can neither be proven nor refuted in our system PE.

By our assumption above, we suppose that γ says that it is not provable in PE.

Now supose for *reductio* that $PE \vdash \gamma$; i.e., γ is a provable in PE. Then since PE is *sound*, γ is true. But γ just says that it is not provable in PE, which is a contradiction. Thus $PE \nvdash \gamma$.

Now suppose for *reductio* that $PE \vdash \neg\gamma$; i.e., $\neg\gamma$ is a theorem of PE. Then since PE is consistent γ is not a theorem of PE. But then since PE is sound, we see that $\neg\gamma$ is true; and this just says that it's not the case that

γ is provable in PE; i.e., γ is provable in PE. But this is a contradiction, so $PE \nvdash \neg\gamma$.

9.2.2. What we need to formalise it. The argument above is very informal. Until we've seen how the detail works, it could be tempting to see this as a piece of *sophistry*. However, it is going to turn out that each of the parts of the argument can be described very precisely; and thus that the whole things works.

However, what we do get from the sketch above it a clear list of things we need to establish:

(1) We need to find a way of representing *provability* using PE;
(2) We need a way of capturing *self-reference* using PE;
(3) From here, we need to make a sentence which *says of itself that it isn't true*.

We are going to spend the rest of this week working on the foundation stone for this problem.

Rather than take on provability and self-reference as separate problems we are going to first show that we can represent the *recursive functions* using PE and that from here we shall make an appeal to Church's thesis to argue that we can represent both provability and self-reference by recursive functions.

Intuitively, the idea is that if we have enough power to represent recursive functions, then the fact that we can do proofs using a pen-and-paper algorithm means that we'll be able to represent all of this activity using our theory.

9.3. A theory of arithmetic - PE

As we've seen before, a theory Γ is a set of sentences which is closed under consequence; i.e., Γ is a theory if for all φ, if $\Gamma \models \varphi$, then $\varphi \in \Gamma$.

Of course, in this form, a theory is a massive collection of sentences. It would clearly be desirable to be able to capture all of this information using something more managable. This is where an axiomatisation come in. In this section, we are going to explore a particular axiomatisation for arithmetic.

Our theory PE is formulated in the language $\mathcal{L}_{Ar} = \{0, s, +, \times, \cdot^{\cdot}\}$. The last symbol here is just a way of capturing the exponentiation function, whic is not usually provided with a symbol.

We then set out the axioms for our theory of arithmetic PE as follows:

(1) $0 \neq sx$
(2) $x \neq y \to sx \neq sy$
(3) $x + 0 = x$
(4) $x + sy = s(x + y)$
(5) $x \times 0 = 0$
(6) $x \times sy = (x \times y) + x$
(7) $x^0 = 1$
(8) $x^{sy} = x^y \times x$
(9) $\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(sx)) \to \forall x\varphi(x)$

REMARK. Each of these axioms employs *implicit universal quantification*. Thus the second axiom, strictly speaking, should be written

$$\forall x \forall y(x \neq y \to sx \neq sy).$$

The final axiom is known as an *axiom schema*. It is actually a representation of infinitely many axioms, each of which is a substitution of a formula (with one free variable) from the language of arithmetic.

9.3.1. Some simple proofs. As we saw in the first weeks of this module, numerals like 2 and 63 are not (strictly) part of the language of arithmetic. However, they are very convenient to use. To facilitate their use we introduce a *metalinguistic function* $\underline{\cdot}$ which takes a number and returns a numeral in the language of arithmetic. So for example, we have:

$$\underline{3} = sss0.$$

We shall also allow this to work for variable so:

$$\underline{\mathrm{n}} = s....s0$$

where there is a sequence of n s's occuring in front of 0.

EXAMPLE 189. $PE \vdash \underline{1} + \underline{1} = \underline{2}$.

Putting this properly into the language we are trying to show that:

$$s0 + s0 = ss0.$$

Here's the proof as a natural deduction:

$$\dfrac{\dfrac{\forall x\forall y(x+sy=s(x+y))}{s0+s0=s(s0+0)} \quad \dfrac{\forall x(x+0=0)}{s0+0=s0}}{s0+s0=ss0}$$

Using full natural deduction will quickly get very cumbersome, so we will actually reason more informally. The effect should be the same.

By identity of terms $s0+s0=s0+s0$. By (4.) we se that $s0+s0=s(s0+0)$, so by the identity rule, we may substitute $s(s0+0)$ for $s0+s0$ wherever we like. Thus $s0+s0=s(s0+0)$. Then by (3.) we have $s0+0=s0$; thus by identity, we have $s0+s0=ss0$.

REMARK. Observe that in the second use of the identity rules, we substituted for a subterm of an expression.

We now use an example which exploits the induction axiom (8.).

EXAMPLE 190. $PE \vdash \forall x(x+0=0+x)$.

We exploit the induction schema by finding an appropriate φ to use. In this case, we simply use the formula (minus the universal quantification) that we are trying to establish. Thus

$$\varphi(n) \leftrightarrow n+0=0+n).$$

Schema (8.) tells us that the following axiom is an axiom of PE:

$$0+0=0+0 \ \wedge \ \forall x(x+0=0+x \rightarrow sx+0=0+sx) \ \rightarrow \forall x(x+0=0+x).$$

So once we have shown the first two parts of the antecedent hold, we can take the consequent of the conditional as our conclusion: which is exactly what we want.

Clear we have $0+0=0+0$ from the logical rules for identity alone. Now take an arbitrary $x \in \omega$ and suppose that $x+0=0+x$. Then

$$\begin{aligned} sx+0 &= sx \\ &= s(0+x) \\ &= 0+sx. \end{aligned}$$

The first $=$ exploited axiom (3.); the second exploited our inductive hypothesis; and the third $=$ exploited axiom (4.). This suffices.

9.3.2. Relating recursion theory back to theories and and logic. Before we get into the hard work, let's take a minute to tie some together the material from this section of the course back to some of the material from the beginning. We'll do this with some interesting definitions for theories.

DEFINITION 191. A theory Γ is *(negation) completene* if for all sentences φ we either:

$$\Gamma \vdash \varphi \text{ or } \Gamma \vdash \neg\varphi.$$

If the goal of our theory is to say everything that we think is true about some subject matter, then negation completeness is clear a must-have. With it, any sentence about the subject matter will be such that either it or its negation will be a theorem. Without it, our theory is incomplete in the sense that there is a sentence which the theory cannot say anything about.

REMARK 192. Observe that since we have soundness and completeness, it doesn't make much difference whether we use $\vdash$ or $\models$. In first order logic, we have discovered that they mean much the same thing.

DEFINITION 193. A theory Γ is *decidable* if the set $\{\varphi \mid \Gamma \vdash \varphi\}$ is recursive.

So we haven't provided a way of representing sentences using natural numbers of Turing tapes, but hopefully by now, you could think of some way of doing this. So let's exploit Church's thesis at this point.

Now decidability is clearly a desirable property too. Given any particular sentence φ, decidability tell us that we can tell whether or not φ is a theorem; i.e., there is an algorithm which will terminate after a finite amount of time telling us that φ is either a theorem of Γ or not.

DEFINITION 194. A theory Γ is *finitely axiomatisable* if there is a finite $\Delta \subseteq \Gamma$ such that for all $\gamma \in \Gamma$, $\Delta \vdash \gamma$.

The value of finite axiomatisability should be clear. It allows us, so to speak, compress a body of knowledge into a finite collection of sentences from which any of the other sentences may be derived. At the very least, it's going to be easier to remember.

Unfortunately, in many interesting cases of theories it is not possible to get a reasonable theory with only finitely many axioms. Such theories are

incomplete, not just in the Gödelian way, but in the sense that they miss out things that are very obviously true.

Fortunately, there is another kind of axiomatisation, which gives, so to speak, a good compression.

DEFINITION 195. A theory Γ is *recursively axiomatisable* if there is a recursive $\Delta \subseteq \Gamma$ such that for all $\gamma \in \Gamma$, $\Delta \vdash \gamma$.

The idea here, is that we there is a recursive function which will tell us whether or not some sentence is an axiom of Γ or not. So we can always figure this out in a finite period of time and then try to work out whether other sentences are theorems of Γ. It's not as good as finitely axiomatisability, but it still gives us more realistic grip on Γ.

9.3.3. Representing the partial recursive functions.

9.3.3.1. *What sort of representation?* Our goal here is to show that for any partial recursive function, there is an arithmetic formula which represents it in the theory PE. Given that we are working with numbers, we shall try to line things up with the Gödel schema approach to algorithms.

Now we don't have any means of constructing new function symbols in PE, but we can build up new formulae which represent relations and sets. For example, the formula:

$$\varphi(x) = \exists y(y \times 2 = x)$$

can be used to represent the even numbers.

So since we know already that functions are just special kinds of relations we shall represent the partial recursive functions with formulae representing their corrseponding relations. So given some recursive function $f : \omega^m \to \omega$, we want a formula in the languge of arithemtic such that $f(\bar{n}) = k$ iff $\varphi(\bar{m}, k)$ is true and we can actually prove that it PE.

The two things we want from our representation are that:

(1) It is *correct*: so the formula doesn't get anything wrong about the function it represents;
(2) PE *can prove* those facts: thus, if it's true then using the proof style above, we can prove it using PE.

REMARK. This actually corresponds to a different kind of soundness and completeness. (1.) asserts that PE is *sound* with respect to the recursive

functions and (2.) asserts that PE is *complete* with respect to the recursive function.

9.3.3.2. *Showing PE is strong enough.* Let us first show that PE can actually prove these facts. To do this, we are actually going to prove something a little stronger than this. We shall do the following:

(1) Define a *natural class of formulae* in the language or arithmetic; then
(2) Show that PE always *gets facts* about such formulae *right*.

In the next section, we shall then show that all of the recursive functions fit into this category so PE does the job.

We now define this natural class of formulae. First of all, observe the following facts about first order logic. First, we observe that it is possible to define the relation $<$ in the language of arithmetic. We have

$$\begin{aligned} x < y &\Leftrightarrow \mathbb{N} \models \exists m(s\underline{\mathbf{x}} + m = \underline{\mathbf{y}}) \\ &\Leftrightarrow \mathbb{N} \models \forall m(\underline{\mathbf{y}} + m \neq s\underline{\mathbf{x}}). \end{aligned}$$

In fact, as you can see, there are a couple of ways of doing it. We use the semantic concept of satisfaction $\models$ here to mean that these facts are true in the standard model of arithmetic, which is what we want. I'll now start using the $<$ symbol as if it were in the language, but on the understanding we could always remove it. [We could also just add a new relation symbol $<$to our language.]

The next thing we need is what is known as *bounded quantification*. This can be used in any model that has an ordering relation on its domain, like the natural numbers $\mathbb{N}$. A bounded quantifer is writen in front of a formula like this

$$\forall x < y \varphi(x).$$

It just means that $\forall x(x < y \to \varphi(x))$, however, it is convenient to cordon of this particular articulation. Similarly, we have a bounded version of existential quantification written

$$\exists x < y \varphi(x)$$

which just means $\exists x(x < y \wedge \varphi(x))$.

We are now ready to define up our natural class of formulae. In fact, we'll define *two of them*. We call the first class the Δ_0 formulae. They are defined as follows:

- if φ is an arithmetic atomic sentence, then φ is Δ_0;
- if φ, ψ are Δ_0, then $\varphi \wedge \psi$, $\neg\varphi$ and $\forall x < y\varphi$ are also Δ_0; and[1]
- nothing else is Δ_0.

We are now going to show that PE is *complete* for sentences which are Δ_0.

THEOREM 196. *Suppose φ is a Δ_0 sentence of arithmetic. Then*

$$\mathbb{N} \models \varphi \;\Rightarrow\; PE \vdash \varphi.$$

REMARK. Observe that we only have one direction $\Rightarrow$ in this theorem. Thus it's probably going to be better to use some kind of positive complexity for this proof.

PROOF. We prove this by induction on the positive complexity of Δ_0 sentences.[2]

(Base) Suppose $\varphi := t = s$ where s and t are arithmetic terms. An induction on the complexity of terms will establish this. Similarly for $\varphi := t \neq s$ where s and t are arithmetic terms. The essential idea is to show that any arithmetic term can be *crunched* back into a numeral using just PE.

(Induction Step) Suppose $\varphi := \psi \wedge \chi$ where ψ and χ are Δ_0 sentences of less positive complexity than φ. Then we have

$$\begin{aligned} \mathbb{N} \models \psi \wedge \chi &\Leftrightarrow \mathbb{N} \models \psi \;\&\; \mathbb{N} \models \chi \\ &\Leftrightarrow PE \vdash \psi \;\&\; PE \vdash \chi \\ &\Leftrightarrow PE \vdash \psi \wedge \chi. \end{aligned}$$

Now suppose $\varphi := \neg(\psi \wedge \chi)$ where $\neg\psi$ and $\neg\chi$ are Δ_0 with less positive complexity than φ. Then

$$\begin{aligned} \mathbb{N} \models \neg(\psi \wedge \chi) &\Leftrightarrow \mathbb{N} \models \neg\psi \text{ or } \mathbb{N} \models \neg\chi \\ &\Rightarrow PE \vdash \neg\psi \text{ or } PE \vdash \neg\chi \\ &\Rightarrow PE \vdash \neg(\psi \wedge \chi). \end{aligned}$$

[1] I'm just going to deal with conjunction, universal quantification and negation so we can cut down on the number of cases that we need to prove below. They can be added back using the usual definitions.

[2] The positive complexity of Δ_0 sentences is the obvious generalisation of definition we used in Week 4.

Now suppose $\varphi := \neg\neg\psi$ where ψ is a $\Delta_0$sentence of less complexity than φ. Then we get

$$\begin{aligned} \mathbb{N} \models \neg\neg\psi &\Leftrightarrow \mathbb{N} \models \psi \\ &\Rightarrow PE \vdash \psi. \end{aligned}$$

Suppose $\varphi := \forall x < \underline{\mathbf{n}}\ \psi(x)$ where $\psi(x)$ is a Δ_0 formula with at most one free variable which has less positive complexity than φ. Then

$$\begin{aligned} \mathbb{N} \models \forall x < \underline{\mathbf{n}}\ \psi(x) &\Leftrightarrow \forall m < n\ \mathbb{N} \models \psi(\underline{\mathbf{m}}) \\ &\Leftrightarrow \mathbb{N} \models \psi(0)\ \&...\&\ \mathbb{N} \models \psi(s...s0) \\ &\Rightarrow PE \vdash \psi(0)\ \&...\&\ \mathbb{N} \models \psi(s...s0) \\ &\Leftrightarrow PE \vdash \psi(0) \wedge ... \wedge \psi(s...s0) \\ &\Leftrightarrow PE \vdash \forall x \leq \underline{\mathbf{n}}\ \psi(x). \end{aligned}$$

The first $\Leftrightarrow$ follows by the satisfaction definition as does the second. At the third stage, we now have sentences so we exploit the induction hypothesis. The rest follows from facts about provability in PE.

Suppose $\varphi := \neg\forall x < \underline{\mathbf{n}}\ \psi(x)$. Then

$$\begin{aligned} \mathbb{N} \models \neg\forall x < \underline{\mathbf{n}}\ \psi(x) &\Leftrightarrow \neg\forall m < n\ \mathbb{N} \models \psi(\underline{\mathbf{n}}) \\ &\Leftrightarrow \mathbb{N} \models \neg\psi(0)\ \text{or...or}\ \mathbb{N} \models \neg\psi(s...s0) \\ &\Rightarrow PE \vdash \neg\psi(0)\ \text{or...or}\ \ PE \vdash \neg\psi(s...s0) \\ &\Leftrightarrow PE \vdash \neg\psi(0) \vee ... \vee \neg\psi(s...s0) \\ &\Leftrightarrow PE \vdash \neg\forall x \leq \underline{\mathbf{n}}\ \psi(x) \end{aligned}$$

□

REMARK 197. Observe that not all of the arrows int he proof above point in both directions. Note that the real trick behind this proof is the fact that we only end up with finite conjunctions and disjunctions, thus there is always a way of checking such facts.

Next we define the Σ_1^0 functions. They are simply the Δ_0 formulae with (possible muliple occurences of) *unbounded* existential quantifiers out the front. So for example, if φ is Δ_0, then $\exists y\varphi$ is Σ_1^0 and so is $\exists z\exists y\varphi$.

However, $\exists y\varphi \wedge \varphi$ is not Σ_1^0 since it is a conjunction of a Δ_1^0 formula and a Σ_1^0 formula. Unfortunately, we are much more likely to come across formulae

of this latter kind than pure Σ_1^0 formulae. But there is a simple way to move between them. We now define this class of formulae and then show how to turn them into Σ_1^0 formulae. We shall call this class Σ formulae. We will rely on these manipulations when we come to represent the process of defining functions by primitive recursion.

We say that:

- if φ is an Δ_0[3], then $\varphi \in \Sigma$;
- if $\varphi, \psi \in \Sigma$ then $\varphi \wedge \psi$ is in Σ;
- if $\varphi \in \Sigma$, then $\neg\neg\varphi, \forall x < y\varphi, \exists x < y\varphi, \neg\forall x\varphi$ and $\exists x\varphi$ are in Σ;
- if $\neg\varphi$ and $\neg\psi$ are in Σ, then $\neg(\varphi \wedge \psi), \neg\forall x < y\varphi, \neg\exists x < y\varphi \in \Sigma$; and
- nothing else is in Σ.

THEOREM 198. *Any Σ formulae can be converted into an equivalent Σ_1^0 formula.*

PROOF. We proceed by induction on the complexity of Σ formulae.

(Base) If φ is Δ_0, then it is clearly Σ_1^0 already.

(Induction Step) Suppose $\varphi := \psi \wedge \chi$. Then by induction hypothesis, there are Σ_1^0 formulae $\exists x\psi_2(x)$ and $\exists y\chi_2(y)$ which are equivalent to ψ and χ respectively. We then let our convervsion of φ be

$$\exists x \exists y(\psi_2(x) \wedge \chi_2(y))$$

which is clearly equivalent to φ.

Suppose $\varphi := \neg\neg\psi$. Then by the induction hypothesis, there is a Σ_1^0 formula $\exists y\psi_2(y)$ which is equivalent (in the standard model of arithmetic) to ψ. Thus, we let our conversion of φ be

$$\exists y \neg\neg\psi_2(y).$$

Suppose $\varphi := \forall x < y\psi(x, y)$. Then by the indution hypothesis, there is a Σ_1^0 formula $\exists z\psi_2(x, y, z)$ which is equivalent to $\psi(x, y)$. We let the following be our conversion

$$\exists w \forall x < y \exists z < w \ \psi_2(x, y, z)$$

and we claim that it is equivalent to φ. We can get φ from the conversion by logic alone. For the other direction, let us take an arbitrary y, then φ tells us that for each $x < y$ there is some, z, let's label it with an x to be clear, z_x,

[3]I.e., φ is either atomic or the negation of an atomic.

such that $\psi_2(x, y, z_x)$. Since there are only finitely many x's there only finitely many z_x's; and thus, there is some w which is greater than all of them.

Thus there is some w such that for every $x < y$ there is a $z < w$ such that $\psi_2(x, y, z)$ which is exactly what our conversion says.

The rest of the cases are left as an exercise. □

THEOREM 199. *PE is Σ_1^0-complete.*

PROOF. Suppose $\varphi := \exists x\psi(x)$ where ψ is Δ_0. Now suppose $\mathbb{N} \models \exists x\psi(x)$. Then there is some n such that $\mathbb{N} \models \psi(\underline{n})$. Moreover since ψ is Δ_0, we have by Theorem 196 that $PE \vdash \psi(\underline{n})$. Thus by (∃-I), we also have $PE \vdash \exists x\psi(x)$. □

9.3.3.3. *Capturing the recursive functions .* So we have now established that our theory PE gets the Σ_1^0 sentences right. Our goal now is to show that we can *represent all of the recursive functions* using Σ_1^0 formulae. We shall focus on representing the Gödel schema. We want to show that any recursive function (represented by an alogorithm of the Gödel schema) can be captured using a formula in the language of arithmetic. Now the *representation* we want works like this.

DEFINITION 200. Let $f : \omega^n \rightharpoonup \omega$ be a partial function. We say that a formula $\psi(x_1, ..., x_n, y) \in Form_{\mathcal{L}_{Ar}}$ *semantically represents* f if

$$f(\langle m_1, ..., m_n\rangle) = k \;\Leftrightarrow\; \mathbb{N} \models \psi(\underline{m_1}, ..., \underline{m_n}, \underline{\mathbf{k}}).$$

So given any partial recursive function $\varphi_e : \omega^n \rightharpoonup \omega$, we want a Σ_1^0 formula $\exists z\psi(z, x_1, ..., x_n, y)$ such that:

$$\varphi_e(\langle m_1, ..., m_n\rangle) = k \;\Leftrightarrow\; \mathbb{N} \models \exists z\psi(z, \underline{m_1}, ..., \underline{m_n}, \underline{\mathbf{k}}).$$

Observe that since φ_e is a partial function, it could well be the case that for some $m_1, ..., m_n$, $\varphi(\langle m_1, ..., m_n\rangle)$ is not defined. The Σ_1^0 aspect of our formula comes in handy here as this will pan out as a value of $m_1, ..., m_n$ for which there is not z such taht $\psi(z, m_1, ..., n_n, k)$ is a truth sentence about the natural numbers.

Now with such a formula in hand, we may then exploit Theorem 199 to go from a the fact that

$$\mathbb{N} \models \exists z\psi(z, \underline{m_1}, ..., \underline{m_n}, \underline{\mathbf{k}})$$

to the fact that

$$PE \vdash \exists z \psi(z, \underline{m_1}, ..., \underline{m_n}, \underline{\mathbf{k}}).$$

Thus, it is in this sense that we have *capturing* the partial recursive function φ_e: if φ_e is defined on some tuple and returns a value, then we there is a sentence of arithmetic witnessing that fact which can be proven in PE.

DEFINITION 201. Let $f : \omega^n \rightharpoonup \omega$. We say that a formula $\varphi \in Form_{\mathcal{L}_{Ar}}$ *captures* f *relative to theory* Γ *if*

$$f(\langle m_1, ..., m_n \rangle) = k \;\Rightarrow\; PE \vdash \psi(\underline{m_1}, ..., \underline{m_n}, \underline{\mathbf{k}}).$$

Thus we capture the partial recursive functions in the sense that for any partial recursive φ_e there is a Σ_1^0 formula $\exists z \varphi(z, x_1, ..., x_n, y \in Form_{\mathcal{L}_{Ar}}$ such that

$$\varphi_e(\langle m_1, ..., m_n \rangle) = k \;\Rightarrow\; PE \vdash \exists z \psi(z, \underline{m_1}, ..., \underline{m_n}, \underline{\mathbf{k}}).$$

So our goal now is to show that the partial recursive functions can be *captured*. However, given Σ_1^0 completeness, it will suffice to merely show that we can semantically represent the partial recursive functions using Σ_1^0 formulae of arithmetic.[4]

We now proceed to do that. As we did last week, we shall represent the basic functions, and then the generators of the Gödel schema.

Let us attempt the z^n function. This is easy. Let $\psi_z(x_1, ..., x_n, y)$ be defined as follows:

$$\psi_z(x_1, ..., x_n, y) \leftrightarrow_d y = 0.$$

Thus no matter what values are used for $x_1, ..., x_n$, the formula is only true if y is 0, which is what we want; i.e., we have:

$$z^n(\langle m_1, ..., m_n \rangle) = k \;\Leftrightarrow\; \mathbb{N} \models \underline{\mathbf{k}} = 0.$$

I'll leave the other two basic functions (successor and projection) as exercises.

We not turn to the rules for *generating* more complex functions:

(1) composition;

[4]Why does this suffice?

(2) primitive recursion; and
(3) minimisation.

I'm only going to do *primitive recursion* here. This is the most involved case. The others are left as exercises. So by definition, we know that given recursive function $f : \omega^m \to \omega$ and $g : \omega^{m+2} \to \omega$ there is another recursive function $h : \omega^{n+1} \to \omega$ such that

$$\begin{aligned} h(0, \bar{n}) &= f(\bar{n}) \\ h(k+1, \bar{n}) &= g(h(k, \bar{n}), k, \bar{n}). \end{aligned}$$

Moreover, we suppose that we have already captured the functions f and g with Σ_1^0 formulae $\varphi(\bar{x})$ and $\gamma(y, z, \bar{x})$ respectively.

Now as we have seen, primitive recursion allows us to run a series of calculation by repeatedly applying the function g to previous outputs. This is why it is called *recursive*. By way of illustration, consider what happens when we put 3 into the general formulation above.

$$\begin{aligned} h(3, \bar{n}) &= g(h(2, \bar{n}), 2, \bar{n}) \\ &= g(g(h(1, \bar{n}), 1, \bar{n}), 2, \bar{n}) \\ &= g(g(g(h(0, \bar{n}), 0, \bar{n}), 1, \bar{n}), 2, \bar{n}) \\ &= g(g(g(f(\bar{n}), 0, \bar{n}), 1, \bar{n}), 2, \bar{n}) \end{aligned}$$

Our goal here is to find a simple way of representing this kind of thing using the language of arithmetic and PE.

To do this, it will be helpful to have a means of representing sequences. To see how this is helpful we might express what is going on in primitive recursion as follows. Considering h as described above we might say:

$h(k, \bar{n}) = m$ iff there is a sequence $\langle b_0, ..., b_k \rangle$ of $k+1$ numbers such that
$f(m) = b_0$ and
for all $x \leq k$, $g(b_x, x, \bar{n}) = b_{x+1}$ and
$b_k = m$

You should see that this *says* the same thing as the definition of h, but we avoid mentioning h on the right hand side. This means we have a *legitimate definition*. Thus if we can represent this statement in arithmetic, then we

can replace any uses of primitive recursion by this instead. This is what we want. We'll finish this off in two stages:

- First, we'll just suppose that we have a way of simply capturing sequences and show that this gives us a *simple way of capturing* h;
- Then we'll show how to *simply capture* sequences.

So let's suppose we have a Σ_1^0 formula $Pro(b, a, m, n)$ which uses both a and b to represent the sequence number and says that the m^{th} value of the sequence (represented by a and b) is n. With this and the formulas φ and γ capturing f and g, we may now capture h using the following formulae ψ, which is just a re-writing of the sentence above into the language of arithmetic:

$$\begin{aligned}\psi(\underline{\mathbf{k}}, \underline{\bar{n}}, \underline{\mathbf{m}}) \quad \equiv \quad & \exists b \exists a(\\ & \exists u < b(Pro(b, a, 0, u) \wedge \varphi(u, \underline{\bar{n}})) \wedge \\ & \forall x < s\underline{\mathbf{k}} \exists w < b \exists u < b(\\ & \quad (Pro(b, a, x, w) \wedge Pro(b, a, sx, u) \wedge \gamma(w, x, \underline{\bar{n}}, u)) \wedge \\ & Pro(a, b, \underline{\mathbf{k}}, \underline{\mathbf{m}})\end{aligned}$$

Since Pro is Σ_1^0, we see by Theorem 198, that ψ is also Σ_1^0, which is what we require.

We now show how to define the Pro formula. So our goal is to represent sequences of number using a pair of numbers a and b. One obvious situation in which we represent sequences of numbers is in our everyday arabic notation. For example, we might think of the number

$$334547$$

as representing the sequence $7, 4, 5, 4, 3, 3$. (It's actually much easier to go backwards, which is why we do so.) This works because:

$$334547 = 7 \times 10^0 + 4 \times 10^1 + 5 \times 10^2 + 4 \times 10^3 + 3 \times 10^4 + 3 \times 10^5.$$

We are going to employ this *trick* to represent arbitrary sequences of numbers. Of course, there is an obvious problem. Using a base of 10 means that we can only represent sequences of numbers $n \leq 9$. But it's not a big problem. To represent sequences of larger numbers we simply need to select a base which is larger than any number in the sequence.

Thus we make use of the following fact.

FACT 202. *For any sequence of numbers $n_0, ..., n_k$ and a such that $a > n_i$ for $1 \leq i \leq m$ there is a unique b such that:*

$$b = n_0 \times a^0 + ... + n_k \times a^k.$$

Since b is unique, it seems reasonable to hope that we might be able to recover each of the elements n_i from the sequence. Indeed this is the case.

To do this we need a couple more simple functions, div and rem. $div(m, n)$ gives the result of dividing m by n but removes any remainder. $rem(m, n)$ gives the remainder that results when m is divided by n. So, for example, $div(14, 5) = 2$ and $rem(14, 5) = 4$.

To get the i^{th} value of the sequence $n_0, ..., n_k$ represented by b with base a we simply take:

$$rem(div(b, a^{i-1}), a).$$

This is just a simple number theoretic fact, but let's see it in action on a simple example. Suppose we are representing the sequence $2, 4, 1, 5$ using the number 5142 in base 10. Then we may recover the 3^{nd} element using the functions above as follows:

$$\begin{aligned} rem(div(5142, 10^{3-1}), 10) &= rem(div(5142, 100), 10) \\ &= rem(51, 10) \\ &= 1 \end{aligned}$$

which is what we wanted.

The last thing we need to do is show how to represent this in the language of arithmetic in a Σ_1^0 way. We leave representing rem and div as an exercise and so we suppose that we have formulae

- $Rem(x, y, z)$ whic says that z is the remainder from dividing x by y; and
- $Quo(x, y, z)$ which says that z is the result of dividing x by y.

We then define $Pro(a, b, m, n)$ as follows:

$$\exists z < b(Quo(b, a^m, z) \wedge Rem(z, a, n).$$

This is clearly Σ_1^0 so we are now done.

REMARK 203. It actually turns out that we can do this in a simpler axiomatisation of arithmetic. It turns out that we can get rid of exponentiation.

Now once we have found the Σ_1^0 formulae which do this, we have basically proven the following two theorems.

THEOREM 204. *If φ is a partial recursive function, then φ may be semantically reprsented by a Σ_1^0 formula.*

PROOF. By induction on the complexity of Gödel schema algorithms, we show that they can be represented by Σ formulae. The base case and inductive steps are shown above. To get a Σ_1^0 formula, we exploit Theorem 198. □

THEOREM 205. *If φ is a partial recursive function, then φ may be captured by PE.*

PROOF. Let φ_e be an arbitrary partial recursive function. Then by Theorme 204,φ_e can be represented by a Σ_1^0 formulae ψ. Then by Theorem 199, we see that ψ will be proven if it is correct. I.e.,

$$\varphi_e(n) = k \;\Leftrightarrow\; \mathbb{N} \models \psi(\underline{\mathbf{n}}, \underline{\mathbf{k}}) \;\Rightarrow\; PE \vdash \psi(\underline{\mathbf{n}}, \underline{\mathbf{k}})$$

□

9.4. Exercises.

EXERCISE 206. Is there a decidable theory which is not negation complete? Is this correct? If so, provide a proof or explanation of this. Otherwise, provide a counterexample. (You may want to use propositional logic rather than first order logic here.)

EXERCISE 207. Is PE finitely axiomatisable, recursively axiomatisable or neither of these?

EXERCISE 208. (*More philosophical*) If PE is recursively axiomatisable, is there something, so to speak, still finite in the way we can represent such a theory.

EXERCISE 209. Complete the other cases in Section 9.3.3.3.

EXERCISE 210. Represent the functions div and rem by Σ_1^0 formula Div and Rem in the language of arithmetic.

EXERCISE 211. Describe a recursive function which lists all the sentence of some language $\mathcal{L}(C)$ which has been appropriately coded. Such a listing could be used to form an enumeration like the one we used to prove the completeness of the natural deduction system.

CHAPTER 10

Incompleteness

10.1. The Diagonal Lemma

We now have the means to show that there is a sentence γ which says of itself that it is not provable.

10.1.1. Gödel coding. First we need a way of talking about sentences of arithmetic using that material of arithmetic: the natural numbers. For this purpose, we use a coding system.

We have seen a number of different coding systems in this course. Let us use the prime decomposition system, which we used to code Turing machines.

Thus we might assign a number to each element of our basic vocabulary as follows:

$\exists$	$\forall$	$\wedge$	$\vee$	$\rightarrow$	$\neg$	$($	$)$	0	s	$+$	$\times$	$\dot{\cdot}$	v	$=$
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

To show how the coding works, we take a simple example.

- $\exists y(y = 2)$.

We then turn this into a proper sentence of $\mathcal{L}_{Ar}$ as follows:

- $\exists v_2(v_2 = ss0)$.

We represent this with the following string of numbers:

- $0, 13, 6, 9, 9, 8, 7, 6, 13, 9, 9, 8, 14, 9, 9, 8, 7$

which we then code using prime decomposition as follows:

- $2^0 \times 3^{13} \times 5^6 \times 7^9 \times 11^9 \times 13^8 \times 17^7 \times 19^6 \times 23^{13} \times 29^6 \times 31^9 \times 37^9 \times 41^8 \times 43^7 \times 47^{14} \times 53^9 \times 59^9 \times 61^8 \times 67^7$.

With a coding system in place, we are now able to express things about sentence in the language of arithmetic.

For formulae of $\mathcal{L}_{Ar}$, we write $\ulcorner\varphi\urcorner$ to denote the code number of φ.

Using similar techniques, we may also code up the proofs of PE too. We shall not detail the process here, although it is also worth considering.

10.1.1.1. *Representation of sets.* We now make more precise the conceptions of representation we need. The first notion is semantic and refers back to the real interpretation of the symbols, while the second is notion is with regard to the ability of PE to prove things.

DEFINITION 212. Let $D \subseteq \omega$. We say that a D is *semantically represented* by some $\varphi(x) \in Form_{\mathcal{L}_{\omega_1\omega}}$ if

$$n \in D \;\Leftrightarrow\; \mathbb{N} \models \varphi(\underline{\mathbf{n}}).$$

REMARK 213. Assuming our theory is consistent, we can make these arrows go in both directions. The important thing is that we are getting the $\neg$ inside the $\Gamma \vdash$ context in the second case; i.e., we aren't just saying that $\Gamma \nvdash \delta(\underline{\mathbf{n}})$ which is much weaker.

THEOREM 214. *All of the recursively enumerable sets are semnatically represented by forrmulae of arithmetic in the standard model.*

PROOF. Let $D \subseteq \omega$ be recursively enumerable. Then there is a partial recursive function φ_e which is defined only on the elements of D; i.e.,

$$n \in D \;\Leftrightarrow\; \varphi_e(n) \text{ is defined.}$$

We observe that for $\varphi_e(n)$ to be defined is just to say that there is a k such that $\varphi_e(n) = k$. Now, we know that all of the partial recursive functions can be semantically represented by formulae of arithmetic; thus, there is some $\psi(x, y) \in Form_{\mathcal{L}_{Ar}}$ such that:

$$\varphi_e(n) = k \;\Leftrightarrow\; \mathbb{N} \models \psi(\underline{\mathbf{n}}, \underline{\mathbf{k}}).$$

Thus we see that

$$n \in D \;\Leftrightarrow\; \mathbb{N} \models \exists y \psi(\underline{\mathbf{n}}, y)$$

and so D is semantically represented by $\exists y \psi(x, y)$. □

COROLLARY 215. *All of the recursive sets are semantically represented by forrmulae of arithmetic.*

DEFINITION 216. Let $D \subseteq \omega$. We say that D is *case-by-case captured* by a theory Γ if:

$$n \in D \;\Rightarrow\; \Gamma \vdash \delta(\underline{\mathbf{n}})$$

and

$$n \notin D \;\Rightarrow\; \Gamma \vdash \neg\delta(\underline{\mathbf{n}}).$$

LEMMA 217. *If $\varphi_e : \omega \rightharpoonup \omega$ is a partial recursive function, then there is a Σ_1^0 formula $\psi(x,y) \in Form_{\mathcal{L}_{Ar}}$ such that*

$$\varphi_e(n) = m \;\Rightarrow\; PE \vdash \forall y(\psi(\underline{\mathbf{n}}, y) \leftrightarrow y = \underline{\mathbf{m}}).$$

Intuitively, this just says that not only can PE capture the function, but it can also show that the formulae which captures φ_e is a genuine function, in the sense of giving unique outputs.

PROOF. (Sketch) Suppose $\varphi_e(n) = m$. We already now that there is some m which PE can prove the existence of. We need to show that PE establishes the uniqueness of this m. We procede by induction (in the metalanguage) on the construction of the formulae we used to represent partial recursive functions which were represented by the Gödel schema. The base case is simple. For example, supposing we had used the formula $\psi(x,y) := x = x \vee y = 0$ to represent z^1 where $z^1(x) = y$. Suppose $z^1(m) = n$. By Theorem9.3.3.3, we already know that $PE \vdash \psi(\underline{\mathbf{m}}, \underline{\mathbf{n}})$. To complete the claim, we must show that $PE \vdash \forall y(y \neq \underline{\mathbf{n}} \rightarrow \neg\psi(\underline{\mathbf{m}}, y)$. To do this, we suppose that there were some arbitrary $a \neq m$ such that $\psi(\underline{\mathbf{n}}, \underline{\mathbf{a}})$ also holds. Simple application of the identity laws show this cannot be. Then universal introduction completes the proof. The rest of the bas case is similar.

To complete the induction step, we go through each of the generation schema and use induction in PE to establish the claim. The proofs for each of these cases is longer than the first case, but involves no further conceptual difficulty. □

THEOREM 218. *PE case-by-case captures all recursive sets.*

PROOF. Suppose $D \subseteq \omega$ is recursive. Then by Exercise187, we know there is a total recursive function $\chi_D : \omega \to \omega$ such that

$$\chi_D(n) = \begin{cases} 0 & \text{if } n \in D \\ 1 & \text{if } n \notin D. \end{cases}$$

By Lemma 217there is a Σ_1^0 formula $\varphi(x, y)$ such that

$$\chi_D(n) = m \;\Rightarrow\; PE \vdash \forall y(\varphi_D(\underline{\mathbf{n}}, y) \to y = \underline{\mathbf{k}}).$$

We now claim that the formula $\varphi_D(x, 0)$ will suffice to capture D; i.e., we shall have

$$\underline{\mathbf{n}} \in D \;\Rightarrow\; PE \vdash \varphi_D(\underline{\mathbf{n}}, 0)$$

and

$$\underline{\mathbf{n}} \notin D \;\Rightarrow\; PE \vdash \neg\varphi_D(\underline{\mathbf{n}}, 0).$$

To see the first of these suppose that $n \in D$. From here we see that $\chi_D(n) = 0$ and by Σ_1^0 completeness, we have $PE \vdash \varphi_D(\underline{\mathbf{n}}, 0)$, which is what we want. On the other hand, let us suppose that $n \notin D$. Then $\chi_D(n) = 1$ and by Σ_1^0 completeness, we have $PE \vdash \varphi_D(\underline{\mathbf{n}}, 1)$. But then since $0 \neq 1$, we may prove from the second fact about φ_D, that $PE \vdash \neg\varphi(\underline{\mathbf{n}}, 0)$, which is what we wanted. □

10.1.1.2. *Provability and substitution.* Let *is-a-proof-of* be the relation between the code of a sentence $\ulcorner\varphi\urcorner$ and the code of a derivation d of that sentence. By Church's thesis, we claim that the is-a-proof-of relation is recursive. Given any pair of a code for a sentence and a code for a derivation, we may verify in a finite amount of time whether or not it proves sentence.

To see this suppose, we were using a Prawitz proof system. We might decode, so to speak, the derivation into its diagramatic form and then verify that every one (of the finitely many) applications of rules was correctly applied. If so, then the pair belongs in the set; otherwise not. This algorithm defines a *total* recursive function which can be made to give the characteristic function of the set of (codes of) sentences and derviations. Thus by Exercise 187, this is a *recursive set*.

Now since the relation is recursive, we may *case-by-case capture* the sentence-derivation pairs using a formula $B_w(x, y)$ such that

$$d \text{ codes a proof of } \varphi \;\Rightarrow\; PE \vdash B_w(x, y)$$

and

$$d \text{ does not code a proof of } \varphi \;\Rightarrow\; PE \vdash \neg B_w(x, y).$$

$B(x, y)$ will be a Σ_1^0 formula as can be seen from the proof of Theorem218 above.

Let *provability* be the property of being the code number of a theorem of PE. With regard to some $\ulcorner\varphi\urcorner$, it says that there is some n such that n is a proof of φ. Since B_w represents a recursive set we can see that the set of codes of provable sentences is recursively enumerable. We claim it is the domain of a partial recursive function.

To see this, suppose again that we are using a Prawitz proof system and that we are trying to verify whether or not φ is provable. We then let our algorithm take codes n of proofs and verify whether or not they are proofs of φ. If they are, then the programme is instructed to halt; otherwise, it keeps looping.

Moreover, we can denote the provability relation by a Σ_1^0 senten $B(x)$ which is such that:

$$B(x) \leftrightarrow \exists n\ B_w(x, n).$$

This is how we we define provability.

Finally, suppose that $x = \ulcorner\varphi(v)\urcorner$ is a formula of arithmetic with at most one free variable. We let $Sub(x, y, z)$ say that z is the code of the formula that results when we *substitute* the numeral for y in the free variable places of the formula coded by x. It should be clear, by Church's thesis that this is a recursive relation and thus we shall assume that $Sub(x, y, z)$ stands for a Σ_1^0 formula of arithmetic. To see this, define an algorithm which takes the code of a formula, a number, and the putative result of a subsitution. The algorithm should check whether the result is correct. It should be clear that the algorithm only need a finite amount of time to verify this, moreover we can make it so that it defines the characteristic function of the set of such triples. Thus this set is recursive.

We summarise these results in the following table:

Set	**type**	Set	$\omega \backslash Set$	**Formula**
$SP = \{\langle\ulcorner\varphi\urcorner, d\rangle \mid d$ codes a proof of φ in $PE\}$	rec	Σ_1^0	Σ_1^0	$B_w(x, y)$
$P\{\ulcorner\varphi\urcorner \mid \varphi$ is provable in $PE\}$	r.e.	Σ_1^0	??	$B(x)$
$Sub\{\langle\ulcorner\varphi(v)\urcorner, y, z\rangle \mid y$ codes the result of substituting y for v in $\varphi(v)$	rec	Σ_1^0	Σ_1^0	$Sub(x, y, z)$

10.1.2. The diagonal lemma. We now show how to make a sentence which says of itself that it is not provable. In fact, we show something stronger.

LEMMA 219. *For any arithmetic formula $\varphi(x)$ there is a sentence γ such that*

$$PE \vdash \gamma \leftrightarrow \varphi^{\ulcorner}\gamma^{\urcorner}.$$

PROOF. Let us define $Diag(x, y)$ to be $Sub(x, x, y)$.

Let $\chi(x) \leftrightarrow_d \exists y(Diag(x, y) \wedge \varphi(y))$.

Let $^{\ulcorner}\chi(x)^{\urcorner} = a$.

Let γ be $\chi(a)$ and $^{\ulcorner}\gamma^{\urcorner} = g$.

Then it should be clear that

$$PE \vdash \forall y(Diag(\underline{\mathbf{a}}, y) \leftrightarrow y = \mathbf{g}).$$

By definition we have

$$PE \vdash \gamma \leftrightarrow \exists y(Diag(\underline{\mathbf{a}}, y) \wedge \varphi(y)).$$

But combining these, we see that

$$PE \vdash \gamma \leftrightarrow \exists y(y = \mathbf{g} \wedge \varphi(y))$$

and so

$$PE \vdash \gamma \leftrightarrow \varphi(g)$$

which, by definition, just means

$$PE \vdash \gamma \leftrightarrow \varphi^{\ulcorner}\gamma^{\urcorner}$$

which is what we wanted to show. □

10.2. Incompleteness

10.2.1. Four theorems.

10.2.1.1. *Incompleteness via soundness (Gödel).*

(Soundness) If $\vdash \varphi$, then $\mathbb{N} \models \varphi$.

This actually entails consistency.

THEOREM 220. *If PE is sound, then there is some sentence γ such that $PE \nvdash \gamma$ and $PE \nvdash \neg\gamma$.*

PROOF. Using Lemma 219, let γ be such that:

$$PE \vdash \gamma \leftrightarrow \neg B^{\ulcorner}\gamma^{\urcorner}.$$

Now suppose for *reductio* that $PE \vdash \gamma$. Then there is a proof of γ in the system PE. Thus we have

$$\mathbb{N} \models B(^{\ulcorner}\gamma^{\urcorner})$$

and since B is Σ_1^0 and PE is Σ_1^0-complete, we have

$$PE \vdash B^{\ulcorner}\gamma^{\urcorner}.$$

But then by our definition of γ we have

$$PE \vdash \neg\gamma$$

and so by soundness we have both $\mathbb{N} \models \gamma$ and $\mathbb{N} \models \neg\gamma$: a contradiction. Thus, $PE \nvdash \gamma$.

Now suppose for *reductio* that $PE \vdash \neg\gamma$. By definition of γ, we then get

$$PE \vdash B^{\ulcorner}\gamma^{\urcorner}.$$

Thus by soundness we have

$$\mathbb{N} \models B^{\ulcorner}\gamma^{\urcorner}$$

and since B semantically represents provability we then have

$$PE \vdash \gamma.$$

Then by soundness again, we get $\mathbb{N} \models \gamma$ and $\mathbb{N} \models \neg\gamma$ which is a contradiction. Thus, $PE \nvdash \neg\gamma$. □

10.2.1.2. *Incompleteness via consistency and ω-consistency (Gödel).*

DEFINITION 221. A theory T in the language of arithmetic is *ω-inconsistent* if $\exists x\varphi(x)$ is in T and for all n, $\neg\varphi(\underline{n})$ is in T too. T is *ω-consistent* if it is not ω-inconsistent.

REMARK 222. This property is related to the existential witnessing property we saw in Week 5.

FACT 223. *ω-consistency implies consistency.*

THEOREM 224. *Suppose PE is ω-consistent. Then there is some sentence γ such that $PE \nvdash \gamma$ and $PE \nvdash \neg\gamma$.*

The proof is much the same as the last one, so we'll have less detail this time.

PROOF. Let γ be such that $PE \vdash \gamma \leftrightarrow \neg B^{\ulcorner}\gamma^{\urcorner}$.

($PE \nvdash \gamma$) Exercise.

($PE \nvdash \neg\gamma$) Suppose for *reductio* that $PE \vdash \neg\gamma$. Then by definition of γ we have

$$PE \vdash B^{\ulcorner}\gamma^{\urcorner}$$

or in other words,

$$PE \vdash \exists x B_w(^{\ulcorner}\gamma^{\urcorner}, x).$$

Now suppose for *reductio* that there was some $n \in \omega$ such that

$$\mathbb{N} \models B_w(^{\ulcorner}\gamma^{\urcorner}, \underline{\mathbf{n}}).$$

Then since B_w semantically represents the proof relation in PE, we have

$$PE \vdash \gamma$$

which is contrary to our initial assumption. Thus there is no such n; i.e., for all n we have

$$\mathbb{N} \models \neg B_w(^{\ulcorner}\gamma^{\urcorner}, \underline{\mathbf{n}})$$

and since B_w case by case captures this relation, we have

$$PE \vdash \neg B_w(^{\ulcorner}\gamma^{\urcorner}, \underline{\mathbf{n}})$$

for all $n \in \omega$. But this, in conjunction with our initial assumption that

$$PE \vdash \exists x B_w(^{\ulcorner}\gamma^{\urcorner}, x)$$

means that PE is ω-inconsistent, which is contrary to our initial assumption. Thus $PE \nvdash \neg\gamma$. □

10.2.1.3. *Incompleteness via consistency (Rosser).*

THEOREM 225. *Suppose PE is consistent. Then there is a sentence ρ such that $PE \vdash \rho$ and $PE \nvdash \neg\rho$.*

PROOF. By Lemma 219, let ρ be such that

$$\rho \leftrightarrow \forall y(B_w(^{\ulcorner}\rho^{\urcorner}, y) \rightarrow \exists z < y\ B_w(^{\ulcorner}\neg\rho^{\urcorner}, z)).$$

The proof that $PE \nvdash \rho$ is similar to the previous proof.

Suppose for *reductio* that $PE \vdash \neg\rho$. Then by definition of ρ, we also have

$$PE \vdash \neg\forall y(B_w(\ulcorner\rho\urcorner, y) \to \exists z < y\ B_w(\ulcorner\neg\rho\urcorner, z)).$$

It should be clear that for some m we have both

$$\begin{aligned} \mathbb{N} &\models B_w(\ulcorner\neg\rho\urcorner, \underline{\mathbf{m}}) \\ \mathbb{N} &\models \forall y(y = \underline{\mathbf{m}} \vee y < \underline{\mathbf{m}} \to \neg B_w(\ulcorner\rho\urcorner, y). \end{aligned}$$

The first follows since B_w semantically represent the proof-of relation and the second follows from the consistency of PE.

Thus we have

$$\begin{aligned} PE &\vdash B_w(\ulcorner\neg\rho\urcorner, \underline{\mathbf{m}}) \\ PE &\vdash \forall y(y = \underline{\mathbf{m}} \vee y < \underline{\mathbf{m}} \to \neg B_w(\ulcorner\rho\urcorner, y)) \end{aligned}$$

by Σ_1^0-completeness. We then see that from the first of these that

$$PE \vdash \forall y(\underline{\mathbf{m}} < y \to \exists z < y\ B_w(\ulcorner\neg\rho\urcorner, z)$$

and by PE alone we have[1]

$$PE \vdash \forall y(\neg(y = \underline{\mathbf{m}} \vee y < \underline{\mathbf{m}}) \to \underline{\mathbf{m}} < y).$$

Contraposing the second or the statements above we get

$$PE \vdash \forall y(B_w(\ulcorner\rho\urcorner, y) \to \neg(y = \underline{\mathbf{m}} \vee y < \underline{\mathbf{m}})).$$

Thus

$$PE \vdash \forall y(B_w(\ulcorner\rho\urcorner, y) \to \exists z < y\ B_w(\ulcorner\neg\rho\urcorner, z));$$

i.e.,

$$PE \vdash \rho$$

which contradicts the consistency of PE. Thus $PE \nvdash \neg\rho$. □

10.2.1.4. *Unprovability of consistency (Gödel).* We now show (less formally) that PE is not capable of proving its own consistency. To do this we look further at the statement of the theorem 225. It says:

- If PE is consistent, then $PE \nvdash \rho$.

[1] Prove this.

But what does consistency mean? It just says that there is no proof from PE of both φ and $\neg\varphi$ for any sentence. However, we can actually express this a little more simply by an equivalent statement. We claim that:

- PE's consistency is equivalent to the statement that $0 = 1$ is not provable from PE.

To see this, let us first go from left to right, but via contraposition. Thus we suppose that $0 = 1$ is provable from PE. Well we also know that $0 \neq 1$ is provable from PE. Thus there is some φ such that both φ and $\neg\varphi$ are provable from PE. For the other direction, let us suppose that for some φ both φ and $\neg\varphi$ are provable from PE. But then we know that any sentence of $\mathcal{L}_{Ar}$ is probable in PE; thus $0 = 1$ is provable in PE.

Thus we shall let $Con(PE)$ be the sentence $\neg\exists n\ B_w(\ulcorner 0 = 1\urcorner, n)$.

But now this puts us in a position to state Theorem 225 in the language of arithmetic. Thus we write:

- $Con(PE) \to \neg B^{\ulcorner}\rho^{\urcorner}$.

Moreover, since B semantically represents provability, we must have:

- $\mathbb{N} \models Con(PE) \to \neg B^{\ulcorner}\rho^{\urcorner}$;

i.e., the corresponding sentence is true in the standard model.

Now it also turns out, although we shall not show it here that:

FACT 226. $PE \vdash Con(PE) \to \neg B^{\ulcorner}\rho^{\urcorner}$.

This means that we can not only express Theorem 225 in the language of arithmetic, but we can actually prove it in PE. The proof of this is beyond our present scope, but the impact of it is very interesting and worth noting. From here, we may simply show that PE cannot prove its own consistency.

THEOREM 227. $PE \nvdash Con(PE)$.

PROOF. Suppose for *reductio* that $PE \vdash Con(PE)$. Then by Fact 226, we have $PE \vdash \neg B^{\ulcorner}\rho^{\urcorner}$. But by the definition of ρ as a diagonal sentence, we also have $PE \vdash \gamma$. But this contradicts Theorem 225; thus, $PE \nvdash Con(PE)$. □

10.2.1.5. *What about other theories?* In this section, we have only considered PE, which might lead us to think that the issue of undecidability can be confined to PE. However, it should be obvious that the arguments that we have used here will be extendable to any theory which is:

- capable of talking about its own syntax; and
- getting the facts about recursive functions right.

10.2.2. Two more theorems. We have seen last week that a theory is decidable if it's theorems form a recursive set. However, we have seen that some sets are recursively enumerable without being recursive. The following definition links the recursion theoretic idea with a logical one.

DEFINITION 228. A theory Γ is *semi-decidable* if Γ is a recursively enumerable.

10.2.2.1. *The set of the theorems of PE are not recursive; i.e., PE is not decidable.*

THEOREM 229. *Suppose PE is consistent. PE is not decidable.*

PROOF. Suppose for *reductio*, that PE is decidable. Then $\Gamma = \{\ulcorner\varphi\urcorner \mid PE \vdash \varphi\}$ is recursive. Since we can capture every recursive function in the language of arithmetic, there must be some (Σ_1^0) formula $B(x)$ such that

$$\ulcorner\varphi\urcorner \in \Gamma \;\Rightarrow\; PE \vdash B\ulcorner\varphi\urcorner$$

and

$$\ulcorner\varphi\urcorner \notin \Gamma \;\Rightarrow\; PE \vdash \neg B\ulcorner\varphi\urcorner.$$

Using the diagonal lemma, let γ be such that

$$PE \vdash \gamma \leftrightarrow \neg B\ulcorner\gamma\urcorner.$$

But then supposing $\gamma \in \Gamma$ we have

$$\begin{aligned} \gamma \in \Gamma &\Rightarrow PE \vdash B\ulcorner\gamma\urcorner \\ &\Leftrightarrow PE \vdash \neg\gamma \\ &\Rightarrow PE \nvdash \gamma \\ &\Rightarrow \gamma \notin \Gamma \end{aligned}$$

which is a contradiction, so $\gamma \notin \Gamma$. (Note that for the second $\Rightarrow$ we appeal to the consistency of PE.) But then we have

$$\begin{aligned} \gamma \notin \Gamma &\Rightarrow PE \vdash \neg B^{\ulcorner}\gamma^{\urcorner} \\ &\Leftrightarrow PE \vdash \gamma \\ &\Leftrightarrow \gamma \in \Gamma \end{aligned}$$

which is another contradiction. Thus Γ is not recursive. □

10.2.2.2. *The set of satisfiable sentences is not recursively enumerable.* We finally return to one of the questions posed early on in the module.

- Is there a way of tweaking the tableau system for first order logic so that we don't need to deal with infinite branches?

In the case of propositional logic, we can certainly do this since we never need to get infinite branches. However, the answer in the case of first order logic is negative.

PROPOSITION 230. *Let $PE - Ind$ be PE without the induction schema. Then $PE - Ind$ is Σ_1^0-complete and can prove Lemma 217.*

PROOF. An inspection of the proof of the Σ_1^0-completeness of PE will show that induction was not used there. More work is required to remove induction from Lemma 217. □

Clearly $PE - Ind$ is incomplete like PE. However, it is also boringly complete. For example, without induction we cannot prove simple facts like

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

so it is not a very useful theory of arithmetic. Nonetheless it is useful for the following reason.

FACT 231. *$PE - Ind$ is finitely axiomatisable.*

PROPOSITION 232. *$PE - Ind$ is not decidable. Moreover, the set*

$$\{^{\ulcorner}\varphi^{\urcorner} \mid PE - Ind \nvdash \varphi\}$$

is not recursively enumerable.

PROOF. The first part is a simple adaptation of the proof of Theorem 229. Indeed, we could have made more general proof there.

For the second part, we need to show that

$$\{\ulcorner\varphi\urcorner \mid PE - Ind \vdash \varphi\}$$

is recursively enumerable. This is left as an exercise. Once we've shown that it is clear that its complement cannot be recursively enumerable too. □

COROLLARY 233. *The set* $V = \{\ulcorner\varphi\urcorner \mid \models \varphi\}$ *is recursively enumerable but not recursive.*

PROOF. The first part can be established by Church's thesis and the methods discussed at the beginning of this week.

For the second part, we shall suppose for *reductio* that $\omega\backslash V$ is recursively enumerable.

Let π be the conjunction of the sentences in $PE - Ind + Tri$.

Let $f : \omega \to \omega$ be the total recursive function which takes codes of sentences $\ulcorner\varphi\urcorner$ and returns $\ulcorner\rho \to \varphi\urcorner$. (We appeal to Church's thesis, but it should be obvious that this is recursive. If not think about what you need to do to get the new code.)

Now consider the set $C = \{\ulcorner\varphi\urcorner \mid \models f(\varphi)\}$.[2] We claim that $\omega\backslash C$ is recursively enumerable. Since we are assuming that $\omega\backslash V$ is recursively enumerable, there is some partial recursive function φ_e whose domain is $\omega\backslash V$. To get a function with domain is $\omega\backslash C$ let ψ be such that

$$\psi(n) = m \;\Leftrightarrow\; \varphi_e(f(n)) = m.$$

This is clearly partial recursive and it domain is clearly $\omega\backslash V$.

But then we see that

$$\begin{aligned} C &= \{\ulcorner\varphi\urcorner \mid \models \rho \to \varphi\} \\ &= \{\ulcorner\varphi\urcorner \mid \vdash \rho \to \varphi\} \\ &= \{\ulcorner\varphi\urcorner \mid PE - Ind + Tri \vdash \varphi\}. \end{aligned}$$

However, we know from Proposition 232, that $\omega\backslash C$ is not recursively enumerable. This is a contradiction, thus $\omega\backslash V$ is not recursively enumerable and V is not recursive. □

[2] My notation is a little sloppy here since we don't put codes of sentences into the context of $\models$. However, the meaning should still be clear.

COROLLARY 234. *The set* $\Sigma = \{\ulcorner\varphi\urcorner \mid \varphi$ *is satisfiable.*$\}$ *(i.e., the set of satisfiable sentences) is not recursively enumerable.*

10.3. Exercises.

EXERCISE 235. Show that ω-consistency implies consistency.

EXERCISE 236. Is there a negation complete theory which is not decidable.

EXERCISE 237. Complete the first half of the proof of Theorem 224.

EXERCISE 238. If PE was not consistent, would it be decidable.