# MATH70132: Mathematical Logic

Lecturer: David Evans

Notes written by Sidharth Hariharan

Imperial College London - Spring 2025

# Contents

# Chapter 1

# Propositional Logic

Propositional logic is the logic of reasoning and proof. Before we get started with anything formal, here's a motivating example.

Consider the following statement:

> If Mr Jones is happy, then Mrs Jones is unhappy, and if Mrs Jones is unhappy, then Mr Jones is unhappy. Therefore, Mr Jones is unhappy.

One can ask ourselves whether it is logically valid to conclude that Mr Jones is unhappy based on the relationship between the happiness of Mr Jones and that of Mrs Jones expressed in the sentence preceding it.

Putting this into symbols, let $P$ denote the statement that Mr Jones is happy, and let $Q$ denote the statement that Mrs Jones is unhappy. We can express the statement as follows:

$$((P \implies Q) \wedge (Q \implies \neg P)) \implies (\neg P) \tag{1.0.1}$$

This disambiguation, by removing any question of marital harmony from what is otherwise a purely logical question, allows us to manually check whether (1.0.1) is a valid statement by constructing a **truth table**.

We will begin by developing some machinery to reason about these sorts of statements more formally.

## 1.1 Propositional Formulae

Broadly speaking, the study of propositional logic involves studying its two major components: **syntax** and **semantics**. While the most formal approach to the study of propositional logic is to study them in that order, in this module, we study semantics before syntax, because while syntax must precede semantics, semantics can serve as motivation for the syntactic choices we make when defining a 'formal system' for propositional logic (see Section 1.2). In this section, we will study the semantics of propositional logic.

### 1.1.1 Propositions and Connectives

We begin by defining the notion of a proposition.

**Definition 1.1.1** (Proposition)**.** A **proposition** is a statement that is either true or false.

**Convention.** We will denote the state of being **true** by **T** and that of being **false** by **F**.

Propositions can be connected to each other using tools known as **connectives**. These can be thought of as **truth table rules**.

**Convention.** Before we define the actual connectives we shall use, we list them down, along with notation.

1. Conjunction ($\wedge$)
2. Disjunction ($\vee$)
3. Negation ($\neg$)
4. Implication ($\rightarrow$)
5. The Biconditional ($\leftrightarrow$)

In particular, we will only use the $\implies$ and $\iff$ symbols when reasoning **informally**. For **formal** use, we will stick to the $\rightarrow$ and $\leftrightarrow$ symbols. In more precise terms, we will use $\implies$ and $\iff$ when reasoning **about** the language we are constructing, whereas we will use $\rightarrow$ and $\leftrightarrow$ when reasoning **within** the language. As we shall see, it will be of paramount importance to distinguish between these two modes of reasoning.

We define them exhaustively as follows.

**Definition 1.1.2** (Connectives). Let $p$ and $q$ be true/false variables. We define each of the connectives listed above to take on truth values depending on those of $p$ and $q$ as follows.

| $p$ | $q$ | $(\neg p)$ | $(\neg q)$ | $(p \wedge q)$ | $(p \vee q)$ | $(p \to q)$ | $(p \leftrightarrow q)$ |
|---|---|---|---|---|---|---|---|
| **T** | **T** | **F** | **F** | **T** | **T** | **T** | **T** |
| **T** | **F** | **F** | **T** | **F** | **T** | **F** | **F** |
| **F** | **T** | **T** | **F** | **F** | **T** | **T** | **F** |
| **F** | **F** | **T** | **T** | **F** | **F** | **T** | **T** |

We are now ready to define the main object of study in this section: propositional formulae.

**Definition 1.1.3** (Propositional Formula). A **propositional formula** is obtained from propositional variables and connectives via the following rules:

(i) Any propositional variable is a propositional formula.

(ii) If $\phi$ and $\psi$ are formulae, then so are $(\neg\phi)$, $(\neg\psi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \to \psi)$, $(\psi \to \phi)$, and $(\phi \leftrightarrow \psi)$.

(iii) Any formula arises in this manner after a finite number of steps.

What this means is that a propositional formula is a string of symbols consisting of

1. variables that take on true/false values,
2. connectors that express the relationship between these variables, and
3. parentheses/brackets that separate formulae within formulae and specify the order in which they must be evaluated when the constituent variables are assigned specific values.

In particular, every propositional formula is either a propositional variable or is built from 'shorter' formulae, where by 'shorter' we mean consisting of fewer symbols.

**Convention.** Throughout this module, we will adopt two important conventions when dealing with propositional formulae.

1. All propositional formulae, barring those consisting of a single variable, shall be enclosed in parentheses.
2. When we want to denote a propositional formula by a certain symbol, we will use the

notation "symobol : formula".

As a concluding remark on the nature of propositional formulae, we will note that just as we use trees to evaluate expressions on the computer when performing arithmetic, we can use them to express and evaluate propositional formulae as well. We will not usually do this, however, as it takes up a lot of space. In any event, we would first need to make precise the notion of *evaluating* a propositional formula. For this, we will turn to the concept of a truth function.

### 1.1.2 Truth Functions

Any assignment of truth values to the propositional variables in a formula $\phi$ determines the truth value for $\phi$ in a **unique** manner, using the exhaustive definitions of the connectives given in Definition 1.1.2. We often express all possible values of a propositional formula in a **truth table**, much like we did in Definition 1.1.2 when defining the connectives.

**Example 1.1.4.** Consider the formula $\phi : ((p \to (\neg q)) \to p)$, where $p$ and $q$ are propositional variables. We construct a truth table as follows.

| $p$ | q | $(\neg q)$ | $(p \to (\neg q))$ | $((p \to (\neg q)) \to p)$ |
|---|---|---|---|---|
| **T** | **T** | **F** | **F** | **T** |
| **T** | **F** | **T** | **T** | **T** |
| **F** | **T** | **F** | **T** | **F** |
| **F** | **F** | **F** | **T** | **F** |

From this table, it is clear that the truth value of $\phi$ depends on the truth values of $p$ and $q$ in some manner (to be perfectly precise, it only depends on the truth value of $p$, and is, in fact, biconditionally equivalent to $p$). We would like to have a formal notion of navigating this dependence to 'compute a value for $\phi$ given values of $p$ and $q$'.

Throughout this subsection, $n$ will denote an arbitrary natural number.

**Definition 1.1.5** (Truth Function). A **truth function** of $n$ variables is a function

$$f : \{\mathbf{T}, \mathbf{F}\}^n \to \{\mathbf{T}, \mathbf{F}\}$$

Before discussing the relevance of truth functions, we will mention a very natural fact.

**Lemma 1.1.6.** *To show two truth functions are equal, it suffices that they take the value* **T** *on precisely the same inputs or that they take the value* **F** *on precisely the same inputs.*

*Proof.* This is obvious, because any truth function can only take one of two values. If they take one value on precisely the same inputs, they must take the other value on the other inputs. This precisely corresponds to what it means for functions to be equal by extensionality. □

These are very directly related to propositional formulae.

**Definition 1.1.7** (Truth Function of a Propositional Formula)**.** Let $\phi$ be a propositional formula whose variables are $p_1, \ldots, p_n$. We can associate to $\phi$ a truth function whose truth value at any $(x_1, \ldots, x_n) \in \{\mathbf{T}, \mathbf{F}\}^n$ corresponds to the truth value of $\phi$ that arises from setting $p_i$ to $x_i$ for all $1 \leq i \leq n$. We define this truth function to be the **truth function of** $\phi$, denoted $F_\phi$.

We can now construct a truth function for the example we saw at the very beginning involving Mr and Mrs Jones (cf. (1.0.1)).

**Example 1.1.8.** `sorry`

We see something quite remarkable here: the truth function of the propositional formula defined in (1.0.1) maps every possible input to **T**! We have a special term for this.

**Definition 1.1.9** (Tautology)**.** A propositional formula $\phi$ is a **tautology** if its truth function $F_\phi$ maps every possible input to **T**.

We can also be more precise about what the biconditional connective actually tells us.

**Definition 1.1.10** (Logical Equivalence)**.** The propositional formulae $\psi$ and $\chi$ are **logically equivalent** if the truth function $F_{\psi \leftrightarrow \chi}$ of their biconditional is a tautology.

We have a fairly basic result about logical equivalence.

**Lemma 1.1.11.** *Let $p_1, \ldots, p_n$ be propositional variables and let $\psi$ and $\chi$ be formulae in $p_1, \ldots, p_n$. Then, $\psi$ and $\chi$ are logically equivalent if and only if $F_\psi = F_\chi$.*

We omit the proof of this result as it merely involves checking things manually. A computer should be able to do this almost instantaneously.

We can also say something about composing formulae together.

**Lemma 1.1.12.** *Suppose that $\phi$ is a propositional formula with variables $p_1, \ldots, p_n$. Let $\phi_1, \ldots, \phi_n$ be propositional formulae. Denote by $\vartheta$ the result of substituting each $p_i$ with $\phi_i$ in $\phi$. Then,*

(i) *$\vartheta$ is a propositional formula.*

(ii) *if $\phi$ is a tautology, so is $\vartheta$.*

(iii) *the truth function of $\vartheta$ is the result of composing the truth function of $\phi$ with the Cartesian product of the truth functions of $\phi_1, \ldots, \phi_n$.*

We do not prove this result either, as it merely involves manual verification.

**Example 1.1.13.** For propositional variables $p_1, p_2$, the statement $(((\neg p_2) \to (\neg p_1)) \to (p_1 \to p_2))$ is a tautology. Therefore, if $\phi_1$ and $\phi_2$ are propositional formulae, then $(((\neg \phi_2) \to (\neg \phi_1)) \to (\phi_1 \to \phi_2))$ is a tautology as well.

We will also mention that a composition being a tautology does not mean the outermost proposition of the composition is a tautology.

**Non-Example 1.1.14.** Let $p$ be a propositional variable. The formula $\phi : (p \to (\neg p))$ is not a tautology. However, we can find a propositional formula $\phi'$ such that $(\phi_1 \to (\neg \phi_1))$ is a tautology: for example, we can define $\phi'$ to be identically **F**.

There are numerous propositional formulae that we know to be logically equivalent. Here is a (non-exhaustive) list.

**Example 1.1.15** (Logically Equivalent Formulae)**.** Let $p_1, p_2, p_3$ be logically equivalent formulae. Then, the following equivalences hold.

1. $(p_1 \wedge (p_2 \wedge p_3))$ is logically equivalent to $((p_1 \wedge p_2) \wedge p_3)$.
2. $(p_1 \vee (p_2 \vee p_3))$ is logically equivalent to $((p_1 \vee p_2) \vee p_3)$.
3. $(p_1 \vee (p_2 \wedge p_3))$ is logically equivalent to $((p_1 \vee p_2) \wedge (p_1 \wedge p_3))$.
4. $(\neg(\neg p_1))$ is logically equivalent to $p_1$.
5. $(\neg(p_1 \wedge p_2))$ is logically equivalent to $((\neg p_1) \vee (\neg p_2))$.
6. $(\neg(p_1 \vee p_2))$ is logically equivalent to $((\neg p_1) \wedge (\neg p_2))$.

Upon inspection, one can find algebraic patterns in the above logical equivalences. There are similarities to the axioms of a **boolean algebra**. We will not explore this further in this module, but we will adopt the convention used in algebra where parentheses are dropped when dealing with associative operations.

**Convention.** We will denote both $(p_1 \wedge (p_2 \wedge p_3))$ and $((p_1 \wedge p_2) \wedge p_3)$ by $(p_1 \wedge p_2 \wedge p_3)$. Similarly, we will denote both $(p_1 \vee (p_2 \vee p_3))$ and $((p_1 \vee p_2) \vee p_3)$ by $(p_1 \vee p_2 \vee p_3)$.

We will end with a combinatorial fact about truth functions.

**Lemma 1.1.16.** *There are $2^{2^n}$ possible truth functions on $n$ variables.*

*Proof.* A truth function is any function from a the set $\{\mathbf{T}, \mathbf{F}\}^n$ to the set $\{\mathbf{T}, \mathbf{F}\}$, with no further restrictions. The former set has $2^n$ elements and the latter set has 2 elements. Therefore, there are $2^{2^n}$ possible truth functions. □

### 1.1.3 Adequacy

We have defined several connectives so far, but we have yet to say anything about whether we will be defining any more connectives going forward. To begin, we will state an important definition.

**Definition 1.1.17** (Adequacy)**.** We say that a set $S$ of connectives is **adequate** if for every $n \geq 1$, every truth function on $n$ variables can be expressed as the truth function as a

propositional formula which only involves connectives from $S$ (and $n$ propositional variables).

The idea that this definition seeks to express is that a set is adequate if and only if for evern $n$, every propositional formula in $n$ variables is logically equivalent to a propositional formula that only contains those $n$ variables and connectives from the set in question. In other words, every propositional formula should admit an equivalent expression that does not contain any connectives apart from those in the set in question. The reason this is expressed in terms of truth functions is that that is how logical equivalence is *defined* (cf. Definition 1.1.10).

We now have the first theorem of this module.

**Theorem 1.1.18.** *The set $\{\neg, \wedge, \vee\}$ is adequate.*

*Proof.* Fix some $n \geq 1$, and let $G : \{\mathbf{T}, \mathbf{F}\}^n \to \{\mathbf{T}, \mathbf{F}\}$ be a truth function. We have two cases.

Case 1. The first case is a trivial case. There are two trivial truth functions on $n$ variables, namely, the constrant truth functions that take the values **T** and **F** for all inputs. Truth is not something encoded 'naturally' into the connectives $\{\neg, \wedge, \vee\}$, but falsity is: the $\neg$ connective directly has to do with expressing falsity. Therefore, the trivial truth function that we will show can always be expressed in terms of the desired connectives is the one that is always false. We show this rigorously.

Assume that $G$ is identically **F**. Then, define the propositional formula $\phi : (p_1 \wedge (\neg p_1))$. Even defining it as a formula on $n$ variables, it is clear to see that its truth function $F_\phi$ is identically **F**. Therefore, $G = F_\phi$.[1]

Case 2. The second case will be the nontrivial case of when a truth function can take on both values **T** and **F**. The way we will show that $\{\neg, \wedge, \vee\}$ is adequate is by constructing a propositional formula in $n$ variables whose truth function is **T** whenever the one in question is **T**. We will do this by isolating the inputs that yield **T** and manipulating propositional variables in a way that corresponds to these inputs.

[1] Admittedly, we are using the Axiom of Extensionality here to define what it means for the two functions to be equal. We will ignore this technicality for now.

Assume that $G$ is not identically $\mathbf{T}$. Then, list all $v \in \{\mathbf{T}, \mathbf{F}\}^n$ such that $G(v) = \mathbf{T}$. Since $\{\mathbf{T}, \mathbf{F}\}^n$ is a finite set, this list is finite, and we can number these $v_1, \ldots, v_r$. For each $1 \leq i \leq r$, denote

$$v_i = (v_{i1}, \ldots, v_{ir})$$

where $v_{ij} \in \{\mathbf{T}, \mathbf{F}\}$ is the $j$th component of $v_i$. Let $p_1, \ldots, p_n$ be propositional variables. Define propositional formulae $(q_{ij})_{1 \leq i \leq r, 1 \leq j \leq n}$ by

$$q_{ij} : \begin{cases} p_j & \text{if } v_{ij} = \mathbf{T} \\ (\neg p_j) & \text{if } v_{ij} = \mathbf{F} \end{cases}$$

Then, $q_{ij}$ has value $\mathbf{T}$ if and only if $p_j$ has value $v_{ij}$. The idea is to now construct a propositional formula that has value $\mathbf{T}$ if and only if $(p_1, \ldots, p_n)$ is one of the $v_i$.

First, we formalise the notion of the $(p_1, \ldots, p_n)$ taking the value of one of the $v_i$. The idea is to combine them using the $\wedge$ connective. Define propositional formulae $(\psi_i)_{1 \leq i \leq r}$ by

$$\psi_i : (q_{i1} \wedge \cdots \wedge q_{in})$$

Then, we have that for all $1 \leq i \leq r$ and $v \in \{\mathbf{T}, \mathbf{F}\}^n$,

$$F_{\psi_i}(v) = \mathbf{T} \iff q_{i1}, \ldots, q_{in} \text{ all have value } \mathbf{T} \iff \text{Each } p_j \text{ has value } v_{ij} \iff v = v_i$$

Next, we combine these $\psi_i$ so that the truth function of the resulting formula is $\mathbf{T}$ if and only if one of the $\psi_i$ is true, a fact that would be equivalent to the input of the truth function being precisely one of the $v_i$. We do this using the $\vee$ connective. Define the propositional formula

$$\vartheta : (\psi_1 \vee \cdots \vee \psi_r)$$

Then, for all $v \in \{\mathbf{T}, \mathbf{F}\}^n$, we have that

$$F_\vartheta(v) = \mathbf{T} \iff \text{One of the } \psi_i \text{ is true} \iff v \text{ is precisely equal to one of the } v_i$$

In partiular, we have that $F_\vartheta(v) = \mathbf{T}$ if and only if $G(v) = \mathbf{T}$ for all $v \in \{\mathbf{T}, \mathbf{F}\}^n$. Then, by Lemma 1.1.6, we are done. □

Before illustrating the point of the above theorem, we make an important definition.

**Definition 1.1.19** (Disjunctive Normal Form)**.** When a propositional formula is expressed only in terms of propositional variables and the set $\{\neg, \wedge, \vee\}$ of connectives, it is said to be in **disjunctive normal form**, which we abbreviate to **DNF**.

What Theorem 1.1.18 then tells us is that every propositional formula is expressible in DNF.

**Corollary 1.1.20.** *For every propositional formula in $n$ variables, there exists a logically equivalent propositional formula in $n$ variables that is in DNF.*

*Proof.* We know that every propositional formula admits a truth function. For any propositional formula in $n$ variables, we can apply Theorem 1.1.18 to its truth function. Then, unfolding the definition of adequacy yields the desired result. □

**Example 1.1.21.** Let $p_1$ and $p_2$ be propositional variables. Consider the propositional formula $\chi : ((p_1 \to p_2) \to (\neg p_2))$. We can see that $F_\chi(v) = \mathbf{T}$ only if $v = (\mathbf{T}, \mathbf{F})$ or $v = (\mathbf{F}, \mathbf{F})$. Therefore, the DNF of $\chi$ is

$$((p_1 \wedge (\neg p_2)) \vee ((\neg p_1) \wedge (\neg p_2)))$$

It turns out that $\{\neg, \wedge, \vee\}$ is not the only adequate set of connectives.

**Example 1.1.22** (Adequate Sets)**.** The following sets of connectives are adequate.

(i) $\{\neg, \vee\}$

(ii) $\{\neg, \wedge\}$

(iii) $\{\neg, \to\}$

The way we can prove this is by simplifying each case using Theorem 1.1.18. Fix propositional variables $p_1, p_2$.

(i) It suffices to show that $p_1 \wedge p_2$ can be expressed using $\neg$ and $\vee$. Indeed,

$$(p_1 \wedge p_2) \text{ is logically equivalent to } (\neg((\neg p_1) \vee (\neg p_2)))$$

(ii) It suffices to show that $p_1 \vee p_2$ can be expressed using $\neg$ and $\wedge$. Indeed,

$$(p_1 \vee p_2) \text{ is logically equivalent to } (\neg((\neg p_1) \wedge (\neg p_2)))$$

(iii) By Case (i), it suffices to show that $p_1 \vee p_2$ can be expressed in terms of $\neg$ and $\rightarrow$. Indeed,

$$(p_1 \vee p_2) \text{ is logically equivalent to } ((\neg p_1) \rightarrow p_2)$$

There are also sets of connectives that are not adequate.

**Non-Example 1.1.23** (Inadequate Sets). The following sets are not adequate.

(i) $\{\wedge, \vee\}$

(ii) $\{\neg, \leftrightarrow\}$

The way we can prove this is by constructing truth functions that cannot be realised by combining propositional variables using only the connectives in the above sets.

(i) No truth function that is identically false can be realised. For that matter, no truth function that maps an input whose every component is **T** to **F** can be realised. Formally, consider any propositional formula $\phi$ built exclusively using a finite set of propositional variables and the connectives $\wedge$ and $\vee$. One can show, by induction on the number of connectives in $\phi$, that $F_\phi(\mathbf{T}, \ldots, \mathbf{T}) = \mathbf{T}$. Since this is true of any $\phi$, a truth function mapping an input of the form $(\mathbf{T}, \ldots, \mathbf{T})$ to **F** is not the truth function of a propositional formula that only includes $\wedge$ and $\vee$.

(ii) No truth function that is identcally true can be realised.

It turns out that there is one connective with a rather astounding adequacy property.

**Definition 1.1.24** (The NOR Connective). Define the **NOR connective**, denoted $\downarrow$, via the following truth table in propositional variables $p$ and $q$.

| $p$ | $q$ | $(p \downarrow q)$ |
|---|---|---|
| **T** | **T** | **F** |
| **T** | **F** | **F** |
| **F** | **T** | **F** |
| **F** | **F** | **T** |

Informally, NOR corresponds to "neither ... nor ...". Formally, we have the following.

**Lemma 1.1.25.** *For all propositional variables $p$ and $q$, the DNF of $(p \downarrow q)$ is given by $((\neg p) \wedge (\neg q))$. In particular, we have that $(p \downarrow q)$ is logically equivalent to $((\neg p) \wedge (\neg q))$.*

We do not write out a proof, as it merely involves comparing truth tables.

**Example 1.1.26** (An Adequate Set with One Connective)**.** It turns out that $\{\downarrow\}$ is connective. Indeed, for propositional variables $p$ and $q$, we have

1. $(p \downarrow p)$ is logically equivalent to $(\neg p)$.
2. $((p \downarrow p) \downarrow (q \downarrow q))$ is logically equivalent to $(p \wedge q)$.

So far, we have been studying *meaning*, in the form of truth functions, but have yet to formally define *what* we are allowed to express that *has* meaning within the propositional paradigm. In other words, we have been studying **semantics** but have yet to define the **syntax** of propositional logic. We will do this in the next section.

## 1.2 A Formal System for Propositional Logic

The motivating idea for everything we shall do in this section is to try and generate *all tautologies* from certain 'basic assumptions', known as **axioms**, using certain **deduction rules**. Together, these will form a **formal system for propositional logic**.

### 1.2.1 Formal Deduction Systems

This subsection gives a very general definition of a formal deduction system, which allows us to construct 'proofs' in a sense more general than propositional logic. We will then specialise this to propositional logic. The material here was **not covered in lectures**, and is based on [**LecNotes2018**].

The first ingredient of a formal system is the set of symbols we are allowed to use within it.

**Definition 1.2.1** (Alphabet)**.** An **alphabet** is a nonempty list of symbols.

For the remainder of this subsection, fix an alphabet $A$. $A$ is not useful on its own: we need to be

able to combine elements of $A$ with each other.

**Definition 1.2.2** (Strings)**.** A **string** is any finite sequence of of elements of $A$.

We do not always want all strings to be useful to us, as we shall see in the case of propositional logic. Our next ingredient in the construction of a formal system is the precise set of strings we are allowed to use.

**Definition 1.2.3** (Formulae)**.** A set of **formulae** is a non-empty subset of a set of strings in $A$.

For the remainder of this subsection, fix a set $\mathcal{F}$ of formulae. It will be important that to distinguish the formulae that will serve as our most basic assumptions and those that will be derived from them.

**Definition 1.2.4** (Axioms)**.** A set of **axioms** is a subset of $\mathcal{F}$.

For the remainder of this subsection, fix a set $\mathcal{A}$ of axioms. We now need to be able to generate formulae from the distinguished ones (ie, axioms).

**Definition 1.2.5** (Deduction Rules)**.** A **deduction rule** is a function that takes in a finite list of formulae in $\mathcal{F}$ and outputs a formula in $\mathcal{F}$.

Together, these ingredients form a **formal deduction system**.

**Definition 1.2.6** (Formal Deduction System)**.** A **formal deduction system** $\Sigma$ is a tuple $(A, \mathcal{F}, \mathcal{A}, \mathcal{D})$, where

1. $A$ is an alphabet
2. $\mathcal{F}$ is a set of formulae in $A$
3. $\mathcal{A}$ is a set of axioms contained in $\mathcal{F}$
4. $\mathcal{D}$ is a set of deduction rules on $\mathcal{F}$

For the remainder of this subsection, fix a formal deduction system $\Sigma = (A, \mathcal{F}, \mathcal{A}, \mathcal{D})$. We can now define what it means to reason in this system.

**Definition 1.2.7** (Proof)**.** A **proof** in $\Sigma$ is a finite sequence of formulae $\phi_1, \ldots, \phi_n \in \mathcal{F}$ such that each $\phi_i$ is either an axiom in $\mathcal{A}$ or is obtained from $\phi_1, \ldots, \phi_{i-1}$ using a deduction rule in $\mathcal{D}$.

We want to be able to isolate the formulae that are 'proven' in this manner.

**Definition 1.2.8** (Theorem)**.** The last formula $\phi_i$ that is contained in a proof $\phi_1, \ldots, \phi_n$ is called a **theorem** of $\Sigma$. We write $\vdash_\Sigma \phi$ to denote that $\phi$ is a theorem of $\Sigma$.

There is a direct correspondence between the way we intuitively think about the proofs and the way we view them here.

**Convention.** We say that proof $P$ in $\Sigma$ is a **proof of a theorem** $\phi$ if $\phi$ is the last formula in the sequence of formulae that make up $P$.

Note the following trivial result.

**Lemma 1.2.9.** *Any axiom is a theorem.*

*Proof.* An axiom $\phi$ is a finite string of formulae consisting of a single formula, namely, itself. Therefore, the proof consisting only of $\phi$ is a proof of $\phi$. □

Finally, we mention a condition on formal deduction systems that lends them to computer-based verification.

**Definition 1.2.10** (Recursive Formal Deduction Systems)**.** Suppose there exists an algorithm that can test whether a string is a formula and whether it is an axiom. Then, $\Sigma$ is called a **recursive formal deduction system**.

The point of the above definition is that in recursive systems, a computer can systematically generate all possible proofs in $\Sigma$ by checking whether every possible formula is an axiom or is derived from axioms using deduction rules. In this case, the validity of any proof can be checked, because each formula in its constituent sequence can be checked.

We now close our discussion on general formal deduction systems. We will now specialise to propositional logic.

### 1.2.2 Constructing a Formal System Propositional Logic

The objective of this subsection is to define the formal system for propositional logic that we will use in this module. We want this formal system to correspond to our intuition in a very specific manner: we would like the theorems in this system to be precisely the tautologies. In other words, we want to construct a system in which theorems are precisely 'statements that are true'.

If we re-examine Definition 1.2.8, we observe something rather interesting: *the definition does not mention truth*! The *validity* of a theorem only has to do with the *deduction rules* of the formal system in which it lives. Therefore, we want to define axioms and deduction rules for propositional logic such that the axioms are 'true' (ie, are *tautologies*, as defined in Definition 1.1.9) and the deduction rules preserve truth. This is an important motivation not only for the choice of deduction rule but also the chocie of (adequate) set of connectives we use in our formal system.

We begin by defining the alphabet.

**Definition 1.2.11** (Alphabet for Propositional Logic)**.** The **alphabet** for propositional logic consists of the following symbols:

1. **Variables** $p_1, p_2, p_3, \ldots$
2. **Connectives** $\neg, \rightarrow$
3. **Punctuation** $(,)$

Next, we define the formulae of propositional logic.

**Definition 1.2.12** (Formulae of Propositional Logic)**.** We define the **formulae** of propositional logic in the following manner.

1. Any variable $p_i$ is a formula.
2. If $\phi$ is a formula, then $(\neg\phi)$ is a formula.
3. If $\phi$ and $\psi$ are formulae, then $(\phi \rightarrow \psi)$ is a formula.
4. Any formula arises from a finite number of applications of the above rules.

Next, we define the axioms of propositional logic. These do appear to be a bit unusual when expressed purely in symbols, but we can interpret them in plain English as well.

**Definition 1.2.13** (Axioms of Propositional Logic)**.** Let $\phi, \psi, \chi$ be formulae in propositional logic. The **axioms** of propositional logic in $\phi$, $\psi$ and (where present) $\chi$ are the following.

(A1) $(\phi \to (\psi \to \phi))$

(A2) $((\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi)))$

(A3) $(((\neg\phi) \to (\neg\psi)) \to (\psi \to \phi))$

We can interpret these axioms as follows.

(A1) If $\phi$ is true, then $\phi$ is implied by any statement.

(A2) If $(\psi \to \chi)$ is true (under some assumption $\phi$), then to prove $\chi$ (assuming $\phi$), it suffices to prove $\psi$ (assuming $\phi$).

(A3) An implication $(\phi \to \psi)$ is implied by its contrapositive $((\neg\phi) \to (\neg\psi))$.

Indeed, we can see that these axioms are all tautologies: they are always true. This is exactly what we want from axioms.

*Remark.* It must be stressed that (A1) and (A2) are axioms *in* $\phi, \psi, \chi$ and (A3) is an axiom *in* $\phi, \psi$. There are therefore infinitely many axioms, one for each choice of $\phi$, $\psi$, and, where applicable, $\chi$. In fact, (A1)-(A3) are sometimes referred to as **axiom schemes** instead of just **axioms** to underscore this.

Finally, we define the sole deduction rule of propositional logic.

**Definition 1.2.14** (Deduction Rule of Propositional Logic)**.** Fix formulae $\phi$ and $\psi$. The **deduction rule** on $\phi$ and $\psi$ states:

(MP) From $\phi$ and $(\phi \to \psi)$, we can deduce $\psi$.

This rule is known as **modus ponens**. We will abbreviate this to (MP).

We can now define the formal deduction system for propositional logic.

**Definition 1.2.15** (Formal Deduction System for Propositional Logic)**.** The **formal deduction system for propositional logic** is the tuple $\mathbf{L} = (A, \mathcal{F}, \mathcal{A}, \mathcal{D})$, where

1. $A$ is the alphabet for propositional logic consisting of countably many propositional variables, as defined in Definition 1.2.11.
2. $\mathcal{F}$ is the set of formulae of propositional logic constructed in terms of the connectives $\neg$ and $\rightarrow$, as defined in Definition 1.2.12.
3. $\mathcal{A}$ is the set of axioms (A1)-(A3) of propositional logic defined in Definition 1.2.13.
4. $\mathcal{D}$ is the deduction rule (MP) defined in Definition 1.2.14.

We are now ready to write formal proofs in **L**.

**Example 1.2.16.** Suppose $\phi$ is any **L**-formula. Then, $(\phi \rightarrow \phi)$ is a theorem of **L**, ie,

$$\vdash_{\mathbf{L}} (\phi \rightarrow \phi) \tag{1.2.1}$$

It is not obvious how to prove this statement using only (A1)-(A3) and (MP). But it is possible. We present a formal proof that consists of a sequence of five formulas, each either axiomatic or deduced from two previous ones using modus ponens.

*Formal proof in* **L**.

1. $(\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi))$
   *Justification: We apply (A1) to* $\phi, (\phi \rightarrow \phi), \phi$.
2. $((\phi \rightarrow ((\phi \rightarrow \phi) \rightarrow \phi)) \rightarrow ((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi)))$
   *Justification: We apply (A2) to* $\phi, (\phi \rightarrow \phi), \phi$.
3. $((\phi \rightarrow (\phi \rightarrow \phi)) \rightarrow (\phi \rightarrow \phi))$
   *Justification: We apply (MP) to steps 2 and 1.*
4. $(\phi \rightarrow (\phi \rightarrow \phi))$
   *Justification: We apply (A1) to* $\phi, \phi, \phi$.
5. $(\phi \rightarrow \phi)$
   *Justification: We apply (MP) to steps 3 and 4.*

□

We need a little more machinery before we can prove that theorems in **L** are precisely the tautologies.

We will develop this in the next subsection.

### 1.2.3 Deductions in L

In this subsection, we will develop some machinery as well as notation to deal with the concept of deduction. However, before we proceed any further, we will give a precise definition of what a deduction is.

Throughout this subsection, fix a set of **L**-formulas $\Gamma$.

**Definition 1.2.17** (Deduction)**.** A **deduction** from $\Gamma$ is a finite sequence of **L**-formulae $\phi_1, \ldots, \phi_n$ such that for all $1 \leq i \leq n$, either

- $\phi_i$ is an axiom of **L**,
- $\phi_i$ is a formula in $\Gamma$, or
- $\phi_i$ is obtained from $\phi_1, \ldots, \phi_{i-1}$ using modus ponens.

Essentially, deductions capture the notion of proofs, with an additional layer of flexibility coming from the fact that we are allowed to have assumptions in $\Gamma$ that go beyond merely the axioms of **L**. Just as proofs end in theorems, deductions end in consequences.

**Definition 1.2.18** (Consequence)**.** We say a formula $\phi$ is a **consequence** of $\Gamma$ if there exists a deduction from $\Gamma$ ending in $\phi$. In this case, we write $\Gamma \vdash_{\mathbf{L}} \phi$. If $\phi$ is not a consequence of $\Gamma$, we write $\Gamma \nvdash_{\mathbf{L}} \phi$.

The relationship between proofs and deductions/theorems and consequences is that the former correspond to the case where $\Gamma$ is empty.

**Convention.** Instead of writing $\emptyset \vdash_{\mathbf{L}} \phi$, we write $\vdash_{\mathbf{L}} \phi$.

Therefore, the theorems in **L** are precisely those formulae that are not consequences of other formulae.

We state a rather trivial fact.

**Lemma 1.2.19.** *Let $\Delta \subseteq \Gamma$. Then, for all* **L***-formulae $\phi$, if $\Delta \vdash_{\mathbf{L}} \phi$, then $\Gamma \vdash_{\mathbf{L}} \phi$.*

We do not prove this fact. In fact, we will often use it without mentioning it explicitly.

There is an important theorem that relates the notion of consequence with that of implication and provides us with a framework of reasoning with statements about consequence.

**Theorem 1.2.20** (The Deduction Theorem)**.** *Let $\phi, \psi$ be* **L***-formulae. Then,*

$$\Gamma \cup \{\psi\} \vdash_{\mathbf{L}} \phi \text{ if and only if } \Gamma \vdash_{\mathbf{L}} (\psi \to \phi)$$

*Proof.* We begin by showing that $\Gamma \cup \{\psi\} \vdash_{\mathbf{L}} \phi$ implies $\Gamma \vdash_{\mathbf{L}} (\psi \to \phi)$.

Assume that $\Gamma \cup \{\psi\} \vdash_{\mathbf{L}} \phi$. This states that there is a deduction from $\Gamma \cup \{\psi\}$ to $\phi$ in the formal system **L**. By Definition 1.2.17, we know that such a deduction must be finite. We prove that for all $n \in \mathbb{N}$, if $\Gamma \cup \{\psi\} \vdash_{\mathbf{L}} \phi$ is a deduction of length $n$, then $\Gamma \vdash_{\mathbf{L}} (\psi \to \phi)$. We proceed by induction on $n$.

Base Case: $n = 1$. We know that $\phi$ is either an axiom or is lies in $\Gamma$ or is $\psi$. In the first two cases, we have

$$\Gamma \vdash_{\mathbf{L}} \phi$$

in which case the axiom (A1) guarantees that

$$\Gamma \vdash_{\mathbf{L}} (\phi \to (\psi \to \phi))$$

Applying (MP) to the deductions above then gives us that

$$\Gamma \vdash_{\mathbf{L}} (\psi \to \phi)$$

as required. In the third case, where $\phi$ is $\psi$, we have

$$\Gamma \vdash_{\mathbf{L}} (\psi \to \psi)$$

by the same argument as in Example 1.2.16.

Inductive Case. sorry

□

From this theorem, we can deduce that the implication connective $\to$ is transitive.

**Corollary 1.2.21** (Hypothetical Syllogism). *Suppose $\phi, \psi, \chi$ are $\mathbf{L}$-formulae such that $\Gamma \vdash_{\mathbf{L}} (\phi \to \psi)$ and $\Gamma \vdash_{\mathbf{L}} (\psi \to \chi)$. Then, $\Gamma \vdash_{\mathbf{L}} (\phi \to \chi)$.*

*Proof.* We show that there is a deduction of $\chi$ from $\Gamma \cup \{\phi\}$. We give a formal proof in $\mathbf{L}$.

1. $\Gamma \vdash_{\mathbf{L}} \phi \to \psi$

   *Justification: Assumption.*

2. $\Gamma \vdash_{\mathbf{L}} \psi \to \chi$

   *Justification: Assumption.*

3. $\Gamma \cup \{\phi\} \vdash_{\mathbf{L}} \psi$

   *Justification: Applying the Deduction Theorem to Step 1.*

4. $\Gamma \cup \{\phi\} \vdash_{\mathbf{L}} \psi \to \chi$

   *Justification: Applying Lemma 1.2.19 to Step 2.*

5. $\Gamma \cup \{\phi\} \vdash_{\mathbf{L}} \chi$

   *Justification: Applying (MP) to Step 4 and Step 3.*

6. $\Gamma \vdash_{\mathbf{L}} \phi \to \chi$

   *Justification: Applying the Deduction Theorem to Step 5.*

The last step is the desired deduction. □

**Example 1.2.22** (A Few Theorems in $\mathbf{L}$). Suppose $\phi$ and $\psi$ are $\mathbf{L}$-formulae. Then,

(a) $\vdash_{\mathbf{L}} ((\neg\psi) \to (\psi \to \phi))$

(b) $\{(\neg\psi), \psi\} \vdash_{\mathbf{L}} \phi$

(c) $\vdash_{\mathbf{L}} (((\neg\phi) \to \phi) \to \phi)$

are all theorems in **L**.

*Proof.*

(a) Problem Sheet 1, Question 6. sorry

(b) We apply (MP) to (a) twice, the first time using our assumption $(\neg\psi)$ and the second time using our assumption $\psi$.

(c) Suppose $\chi$ is any formula. Then, from (b), we can see that

$$\{(\neg\phi), ((\neg\phi) \to \phi)\} \vdash_{\mathbf{L}} \chi \tag{1.2.2}$$

because we would be able to apply (MP) to the assumptions to deduce both $\phi$ and $(\neg\phi)$.

Now, let $\alpha$ be an axiom and let $\chi$ be $(\neg\alpha)$. Then, (1.2.2) tells us that

$$\{(\neg\phi), ((\neg\phi) \to \phi)\} \vdash_{\mathbf{L}} (\neg\alpha)$$

sorry

□

## 1.3 Important Properties of L

In this section, we prove important properties about **L**, with our first major goal being to prove that the theorems in **L** are precisely the tautologies.

### 1.3.1 Propositional Valuations

**Definition 1.3.1** (Propositional Valuation). A **propositional valuation v** is an assignment of truth values to propositions $p_1, p_2, \ldots$, so that

$$\mathbf{v}(p_i) \in \{\mathbf{T}, \mathbf{F}\}$$

for all $i \in \mathbb{N}$.

Using the truth table rules by which we defined the connectives $\to$ and $\neg$ of **L** (cf. Definition 1.1.2), we can assign a truth value $\mathbf{v}(\phi)$ to any **L**-formula $\phi$.

**Lemma 1.3.2** (Behaviour of Propositional Valuations). *Let $\phi, \psi$ be* **L***-formulae. Then,*

1. $\mathbf{v}((\neg\phi)) \neq \mathbf{v}(\phi)$
2. $\mathbf{v}(\phi \to \psi) = \mathbf{F}$ *if and only if* $\mathbf{v}(\phi) = \mathbf{T}$ *and* $\mathbf{v}(\psi) = \mathbf{F}$

We do not prove these results.

We can now be precise about what a tautology is.

**Definition 1.3.3** (Tautology). An **L**-formula $\phi$ is a **tautology** if $\mathbf{v}(\phi) = \mathbf{T}$ for all propositional valuations $\mathbf{v}$.

We are now ready to prove that **L** is sound.

### 1.3.2 Soundness

**Definition 1.3.4** (Soundness). We say a formal system is **sound** if every theorem in it is a tautology.

Now, the much-awaited result.

**Theorem 1.3.5** (Soundness of **L**). *The formal system* **L** *of propositional logic is sound.*

*Proof.* Let $\phi$ be a theorem in **L**. We prove that $\phi$ is a tautology by performing induction on the length of $\phi$. The base case involves proving that the axioms (A1)-(A3) are tautologies, and the inductive case involves proving that the modus ponens deduction rule (MP) preserves tautologies.

While these proofs can be done manually, there is a rather clever trick that can be used in the case of (A2). This is given in [**LecNotes2018**]. `sorry` □

We have a more general version of this result.

**Theorem 1.3.6** (A Generalisation of Soundness). *Let $\Gamma$ be a set of* **L***-formulae, and let $\phi$ be an* **L***-formula such that $\Gamma \vdash_{\mathbf{L}} \phi$. If $\mathbf{v}(\psi) = \mathbf{T}$ for all $\psi \in \Gamma$, then $\mathbf{v}(\phi) = \mathbf{T}$.*

*Proof.* The proof is actually identical to that of Theorem 1.3.5, but with $\Gamma$ being a part of the base case. In this case, the proof follows from the assumption that $\mathbf{v}$ is true on $\Gamma$. □

Now that we have shown that $\mathbf{L}$ is sound, we show that $\mathbf{L}$ has other important properties that will enable us to reason in $\mathbf{L}$ the way we would like to.

### 1.3.3 Consistency

For the remainder of this section, fix a set $\Gamma$ of $\mathbf{L}$-formulae.

**Definition 1.3.7** (Consistency). $\Gamma$ is **consistent** if there is no $\mathbf{L}$-formula $\phi$ such that $\Gamma \vdash_{\mathbf{L}} \phi$ and $\Gamma \vdash_{\mathbf{L}} (\neg\phi)$.

Soundness tells us something about the consistency of the empty set in $\mathbf{L}$, which we will refer to more generally as the **consistency of $\mathbf{L}$**.

**Theorem 1.3.8** (Consistency of $\mathbf{L}$). *There is no $\mathbf{L}$-formula $\phi$ such that $\vdash_{\mathbf{L}} \phi$ and $\vdash_{\mathbf{L}} (\neg\phi)$.*

*Proof.* This follows from Theorem 1.3.5: if such an $\mathbf{L}$-formula $\phi$ did exist, both $\phi$ and $(\neg\phi)$ would need to be tautologies, violating Lemma 1.3.2. Therefore, such an $\mathbf{L}$-formula cannot exist. □

We have a more general result.

**Proposition 1.3.9.** *Suppose $\Gamma$ is a consistent set of $\mathbf{L}$-formulae. Let $\phi$ be an $\mathbf{L}$-formula that is* not *a consequence of $\Gamma$. Then, $\Gamma \cup \{(\neg\phi)\}$ is consistent.*

*Proof.* Suppose that $\Gamma \cup \{(\ neg\phi)\}$ is not consistent. Then, there exists a formula $\psi$ such that

$$\Gamma \cup \{(\neg\phi)\} \vdash_{\mathbf{L}} \psi \tag{1.3.1}$$

$$\Gamma \cup \{(\neg\phi)\} \vdash_{\mathbf{L}} (\neg\psi) \tag{1.3.2}$$

We can apply the Deduction Theorem (Theorem 1.2.20) to (1.3.2) to deduce that

$$\Gamma \vdash_{\mathbf{L}} ((\neg\phi) \to (\neg\psi)) \tag{1.3.3}$$

Then, applying (A3) to (1.3.3), we have

$$\Gamma \vdash_{\mathbf{L}} (\psi \to \phi) \tag{1.3.4}$$

In similar fashion, we can apply the Deduction Theorem followed by (A3) to (1.3.1) to deduce that

$$\Gamma \vdash_{\mathbf{L}} ((\neg\psi) \to \phi) \tag{1.3.5}$$

sorry □

### 1.3.4 Completeness

We begin by making an observation about **L**: in general, it is not true that for any given **L**-formula, either it or its negation is a theorem.

**Definition 1.3.10** (Completeness). Let $\Gamma$ be a consistent set of **L**-formulae. We say $\Gamma$ is **complete** if for all **L**-formulae $\phi$, either $\Gamma \vdash_{\mathbf{L}} \phi$ or $\Gamma \vdash_{\mathbf{L}} (\neg\phi)$.

It turns out that all consistent sets of **L**-formulae are contained in complete sets of formulae.

**Theorem 1.3.11** (Lindenbaum's Lemma). *Suppose $\Gamma$ is a consistent set of* **L***-formulae. Then, there exists a complete set of* **L***-formulae $\Gamma^*$ such that $\Gamma \subseteq \Gamma^*$.*

*Proof.* The idea is to construct $\Gamma^*$ by brute force. We know that the alphabet of **L** is countable. Therefore, so is the set of all possible **L**-formulae. We can list them as $\phi_0, \phi_1, \ldots$. We perform the fomllowing inductive construction: let $\Gamma_0 := \Gamma$ and for all $n \geq 0$, define

$$\Gamma_{n+1} := \begin{cases} \Gamma_n & \text{if } \Gamma_n \vdash_{\mathbf{L}} \phi_n \\ \Gamma_n \cup \{(\neg\phi_n)\} & \text{if } \Gamma_n \nvdash_{\mathbf{L}} \phi_n \end{cases} \tag{1.3.6}$$

One can perform induction on $n$ to show that for all $n \geq 0$,

- $\Gamma_n \subseteq \Gamma_{n+1}$ (using purely (1.3.6))
- $\Gamma_n$ is consistent (using Proposition 1.3.9)

Define

$$\Gamma^* := \bigcup_{n=0}^{\infty} \Gamma_n$$

Before we can show that $\Gamma^*$ is complete, we need to show that it is consistent: see Definition 1.3.10. To that end, suppose there exists some $\mathbf{L}$-formula $\phi$ such that $\Gamma^* \vdash_{\mathbf{L}} \phi$ and $\Gamma^* \vdash_{\mathbf{L}} (\neg\phi)$. This means there is a finite sequence of deductions from $\Gamma^*$ and the axioms of $\mathbf{L}$ using only (MP) that ends in $\phi$ and another one that ends in $(\neg\phi)$. There exist $i, j \in \mathbb{N}$ such that these sequences are contained in $\Gamma_i$ and $\Gamma_j$ respectively, because the $\Gamma_n$s represent all possible deductions that can be made from $\Gamma$ (and the axioms). Since we have a chain of inclusions, we have either $\Gamma_i \subseteq \Gamma_j$ or $\Gamma_j \subseteq \Gamma_i$. In either case, we have a contradiction, because $\Gamma_i$ and $\Gamma_j$ are both consistent, but we would be able to deduce both $\phi$ and $(\neg\phi)$ from one of them. Therefore, $\Gamma^*$ is consistent too.

Finally, we show that $\Gamma^*$ is complete. Let $\phi$ be a formula. By the enumeration above, we know that $\phi = \phi_n$ for some $n \in \mathbb{N}$. We know either $\Gamma_n \vdash_{\mathbf{L}} \phi_n$ or $\Gamma_n \nvdash_{\mathbf{L}} \phi_n$. If the former is true, we have that $\Gamma^* \vdash_{\mathbf{L}} \phi_n$ and we are done. Else, by our construction in (1.3.6), we have that $\Gamma_{n+1} \vdash_{\mathbf{L}} (\neg\phi)$, and therefore, $\Gamma^* \vdash_{\mathbf{L}} (\neg\phi)$. Therefore, from $\Gamma^*$, we can deduce either $\phi$ or $(\neg\phi)$, making $\Gamma^*$ complete, as required. □

Going forward, we will adopt the following notation.

**Convention.** Let $\Gamma$ be a consistent set of $\mathbf{L}$-formulae. We will denote by $\Gamma^*$ the complete set of $\mathbf{L}$-formulae containing $\Gamma$ as given in Theorem 1.3.11.

It turns out that the construction allows us to prove the existence of an important kind of valuation.

**Proposition 1.3.12.** *Let $\Gamma$ be a consistent set of $\mathbf{L}$-formulae. Then, there exists a propositional valuation $\mathbf{v}$ such that*

$$\mathbf{v}(\phi) = \mathbf{T} \text{ if and only if } \Gamma^* \vdash_{\mathbf{L}} \phi$$

*Proof.* `sorry` □

# Chapter 2

# First-Order Logic

The plan for this part of the module is to examine Predicate Logic, or First-Order Logic. We will examine the following, though the distinction between semantics and syntax shall not be as pronounced as it is below:

1. Semantics:
    (a) First-order structures
    (b) First-order languages and the corresponding formulae
2. Syntax:
    (a) A formal system for first-order logic
    (b) Gödel's Completeness Theorem, which tells us that the theorems of the formal system are the "logically valid formulae"

Before we can begin the study of first-order logic, we need to make several important definitions and introduce the notation we will use for the remainder of this chapter. We will begin by studying structures and first-order languages. We will then relate ideas from propositional logic to ideas in first-order logic. Finally, we will build a formal system for first-order logic and give a rigorous syntactic foundation for the ideas we discuss.

## 2.1 Languages, Structures and Interpretations

In this section, we discuss the notion of first-order languages and their interpretations in first-order structures. While this is primarily a study of semantics, the definition of languages is syntactic in nature. Yet, we consider this section to be a study of semantics because the purpose is to give some sort of meaning to the syntactic expressions we have in first-order languages.

We begin by studying first-order structures in an abstract sense. We then take a syntactic detour into the study of first-order languages, before examining what it means for structures to exist inside (and give interpretations for) first-order languages.

### 2.1.1 First-Order Structures

We begin by discussing relations and functions of a given arity.

**Definition 2.1.1** ($n$-ary Relation on a Set)**.** Suppose $A$ is a set and $n \geq 1$ is a natural number. An **$n$-ary relation on** $A$ is a subset

$$\overline{R} \subseteq \{(a_1, \ldots, a_n) \mid a_1, \ldots, a_n \in A\}$$

We have a similar notion for functions, with the key fact being that $n$-ary functions take in $n$ inputs and return a single output, and all inputs and outputs must come from the set in question.

**Definition 2.1.2** ($n$-ary Function on a Set)**.** Given a set $A$, an **$n$-ary function on** $A$ is a function

$$\overline{f} : A^n \to A$$

We make a subtle distinction between functions and relations in formal and informal language. This is something that will get clearer as we progress.

**Convention.** The reason why we put bars on top of the symbols is to distinguish functions and relations as they appear in formulae from the way that discuss them.

We have special terms when $n = 1, 2, 3$.

**Convention.**

1. A 1-ary relation is commonly called a **unary relation**.
2. A 2-ary relation is commonly called a **binary relation**.
3. A 3-ary relation is commonly called a **ternary relation**.

These notions are not new to us.

**Example 2.1.3** (Some Familiar $n$-ary Relations).

1. Equality is a binary relation on any set.
2. $\leq$ is a binary relation on $\mathbb{R}$.
3. $\{x \in \mathbb{Z} \mid x \text{ is even}\}$ is a unary relation on $\mathbb{Z}$.

Admittedly, the fact that the third example is precisely a set is a little unusual to see. This is because in practice, the following convention is used.

**Convention.** Let $\overline{R} \subseteq A^n$ be a relation on some set $A$. For all $(a_1, \ldots, a_n) \in A^n$, when we write

$$\overline{R}(a_1, \ldots, a_n)$$

or say that

$$\overline{R}(a_1, \ldots, a_n) \text{ holds}$$

we mean that $(a_1, \ldots, a_n) \in \overline{R}$.

We are now ready for the most important definition of this chapter.

**Definition 2.1.4** (First-Order Structure). A **first-order structure** is the following data:

1. a non-empty set $A$ called the **domain** of $\mathcal{A}$.
2. a set of **relations** on $A$

$$\left\{ \overline{R_i} \subseteq A^{n_i} \;\middle|\; i \in I \right\}$$

3. a set of **functions** on $A$

$$\left\{\overline{f_j} : A^{m_j} \to A \;\middle|\; j \in J\right\}$$

4. a set of **constants** that are elements of $A$

$$\{\overline{c_k} \in A \mid k \in K\}$$

where $I, J, K$ are index sets that can be empty.

Usually, the index sets of a first-order structure are subsets of $\mathbb{N}$, but in principle, they could be any set. We package the information about the constants and the arity of the functions and relations together in the following manner.

**Definition 2.1.5** (Signature). Let $\mathcal{A}$ be a first-order structure. The **signature** of $\mathcal{A}$ is the information

$$\{n_i \mid i \in I\} \qquad \{m_j \mid j \in J\} \qquad K$$

with the respective sets describing the arity of the relations on $A$, the arity of the functions on $A$, and the index set of the constants in $A$.

We use the following notation for first-order structures.

**Convention.** For a first-order structure $\mathcal{A}$ given as above, we wil deonte

$$\mathcal{A} = \left\langle A; \{\overline{R_i}\}_{i\in I}, \{\overline{f_j}\}_{j\in J}, \{\overline{c_k}\}_{k\in K} \right\rangle$$

More generally, we use the notation

$$\text{Structure} = \langle \text{Domain; Relations, Functions, Constants} \rangle$$

Often, we even drop the { } when describing first-order structures: for instance, we would simply write

$$\langle \mathbb{Z}; =, +, -, 0 \rangle$$

as opposed to

$$\langle \mathbb{Z}; \{=\}, \{+, -\}, \{0\} \rangle$$

to describe the group of integers along with the relation of equality, the binary function of addition, the unary function of inversion by sign chance, and the identity 0.

We have encountered any number of first-order structures so far. Here are a few examples.

The first is a very basic example.

**Example 2.1.6** (Orderings). We can take $A$ to be one of the sets $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$. We can define a first-order structure on $A$ with only one unary relation—that of ordering—and no functions or constants.

It is important to note that while the sets $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ all admit richer structures on them, they are not needed to define ordering. We don't even include equality in this description because a formula that contains an equality symbol is not about ordering.

In the next example, we look at an algebraic structure.

**Example 2.1.7** (Groups). Every group is a first-order structure with the following data.

1. The domain is the set of elements of the group.
2. The sole relation is the binary relation of equality.
3. There is a binary function for the group operation and a unary function for inversion.
4. There is a constant for the identity element.

We can make a similar definition for rings.

We do not even need to talk about objects that we usually deal with as sets. We can also talk about graphs, which, while defined in terms of sets, are usually studied visually.

**Example 2.1.8** (Graphs). Graphs (or, more precisely, their vertices), along with two binary relations—equality and adjacency—and no functions or constants, form a first-order structure.

### 2.1.2 First-Order Languages

We are now ready to formally define the notion of first-order languages.

**Definition 2.1.9** (First-Order Language). A **first-order language** $\mathcal{L}$ consists of the following data.

1. Index sets $I, J, K$ where $I$ is non-empty but $J$ and $K$ can be empty.
2. An **alphabet** of symbols, consisting of
   (a) **Variables** $x_0\ x_1\ x_2\ \cdots$
   (b) **Connectives** $\neg\ \rightarrow$
   (c) **Punctuation** $(\ )\ ,$
   (d) The **Quantifier** $\forall$
   (e) **Relation symbols** $R_i$ for $i \in I$
   (f) **Function symbols** $f_j$ for $j \in J$
   (g) **Constant symbols** $c_k$ for $k \in K$
3. An **arity** for every relation and function symbol.

The arity and cardinality information of a first-order language is encoded in the following manner.

**Definition 2.1.10** (Signature). Let $\mathcal{L}$ be a first-order language with index sets $I, J, K$ such that $I$ is non-empty and

1. The relations $\{R_i \mid i \in I\}$ have arities $\{n_i \mid i \in I\}$
2. The functions $\{f_j \mid j \in J\}$ have arities $\{m_j \mid j \in J\}$
3. The constants are given by $\{c_k \mid k \in K\}$

The information

$$\{n_i \mid i \in I\} \qquad \{m_j \mid j \in J\} \qquad K$$

is called the **signature** of $\mathcal{L}$.

In principle, one should very precisely define punctuation rules and the general notation for expressing formulae in a first-order language. Indeed, what this means is that one should define what it means for a string of symbols to be 'well-formed'. This is somewhat laborious, so we simply adopt the following convention.

**Convention.** We use the punctuation symbols

$$(\quad)\quad,$$

in the following manner.

- We enclose all expressions involving connectives in parentheses, barring those involving a single variable or a single constant.
- We enclose statements of the form "$\forall x$" in parentheses.
- We denote applications functions $f$ by $f(\cdots)$ and relations $R$ by $R(\cdots)$.
- We use commas to separate the arguments of functions and relations.

There are numerous symbols in first-order languages. It makes sense to isolate the ones that form the 'objects' with which we 'reason' in this language.

**Definition 2.1.11** (Terms). Let $\mathcal{L}$ be a first-order language. The set of **terms** of $\mathcal{L}$ is the smallest set such that

1. Every variable is a term.
2. Every constant is a term.
3. If $t_1, \ldots, t_n$ are terms and $f$ is an $n$-ary function symbol, then $f(t_1, \ldots, t_n)$ is a term.

Moreover, we shall stipulate that every term arises in this manner.

We can define very basic first order languages.

**Example 2.1.12.** Let $\mathcal{L}$ be a first-order language such that

1. There are no relations
2. There is a binary function $f$
3. There are two constants $c_1, c_2$

Some terms of $\mathcal{L}$ are

$$c_1 \quad c_2 \quad x_1 \quad f(c_1, c_2) \quad f(x_1, c_1) \quad f(x_1, f(c_1, c_2)) \quad f(f(c_1, x_1), c_2)$$

There are many other terms. Note that we automatically assumed the existence of variables $x_1, x_2, \ldots$, which is consistent with Definition 2.1.9.

**Non-Example 2.1.13.** Let $\mathcal{L}$ be the first-order language given in Example 2.1.12. A string of symbols from the alphabet of $\mathcal{L}$ that is **not** a term is

$$ff x_1$$

The reason for this is that $x_1$ is not applied to $f$, and even if we were to ignore the punctuation convention of writing function arguments inside parentheses, we would have that the arity of $f$ is violated, because a function of two variables is being applied (twice) to a single input (or the leftmost $f$ is being applied to both $f$ and $x_1$, which contradicts the fact that $f$ takes inputs that are both terms, and $f$ alone is not a term). It is precisely to avoid ambiguities of this sort that we have punctuation conventions; either way, in this case, there are too many errors for $ff x_1$ to be a term in $\mathcal{L}$.

We now define a way of using the quantifiers and connectives of first-order language s to build *formulae*. The idea is to define a fundamental notion of formulae using purely the relations of the language and then define how more complex formulae can be built from them.

**Definition 2.1.14** (Atomic Formula)**.** Let $\mathcal{L}$ be a first-order language. An **atomic formula** of $\mathcal{L}$, or an $\mathcal{L}$-**atomic formula**, is an expression of the form

$$R(t_1, \ldots, t_n)$$

where $R$ is an $n$-ary relation symbol in $\mathcal{L}$ and $t_1, \ldots, t_n$ are terms.

Note that in the above definition, we do not require the inputs $t_1, \ldots, t_n$ to be variables or constants. We merely require them to be *terms*. This means they could be variables, constants, or outputs of functions applied to other terms. Being 'atomic' has only to do with being the output of a relation symbol. We can now define what a formula is in a broader sense.

**Definition 2.1.15** (Formula)**.** Let $\mathcal{L}$ be a first-order language. The **formulae** of $\mathcal{L}$, or the $\mathcal{L}$-**formulae**, are defined as follows.

1. Every atomic formula is a formula.
2. If $\phi$ is a formula, then so is $(\neg\phi)$.

3. If $\phi$ and $\psi$ are formulae, then so is $(\phi \to \psi)$.
4. If $\phi$ is a formula, then so is $(\forall x)\,\phi$.

Moreover, we stipulate that every formula arises in this manner.

We can define very simple formulae in very simple first-order language s.

**Example 2.1.16.** Let $\mathcal{L}$ be a first-order language with

1. One unary relation symbol $P$ and one binary relation symbol $R$
2. One binary function symbol $f$
3. Two constants $c_1$ and $c_2$

Then, the following are all atomic formulae:

$$P(x_1) \quad R(c_1, x_1) \quad R(f(x_1, c_1), c_2)$$

Similarly, the following are all formulae:

$$\neg P(x_1) \quad (P(x_1) \to R(c_1, x_1)) \quad (\forall x)\,R(x, c_1)$$

There is a reason why we only allowed first-order language s to have connectives $\to$ and $\neg$ and quantifier $\forall$: we can build the other connectives $(\wedge, \vee, \leftrightarrow, \ldots)$ and the other quantifier $(\exists)$ from these.

**Definition 2.1.17** (The Existential Quantifier)**.** Let $\mathcal{L}$ be a first-order language and let $\phi$ be an $\mathcal{L}$-formula. Then, we define

$$(\exists x)\,\phi$$

to be shorthand for the formula

$$(\neg\,(\forall x)\,(\neg\phi))$$

We also define the other connectives as in propositional logic.

**Definition 2.1.18** (Connectives). Let $\mathcal{L}$ be a first-order language. Let $\phi$ and $\psi$ be $\mathcal{L}$-formulae. We define the connectives $\wedge, \vee, \leftrightarrow, \uparrow, \downarrow$ as follows:

$$(\phi \wedge \psi) \text{ is shorthand for } (\neg(\phi \to \neg\psi))$$
$$(\phi \vee \psi) \text{ is shorthand for } ((\neg\phi) \to \psi)$$
$$(\phi \leftrightarrow \psi) \text{ is shorthand for } ((\phi \to \psi) \wedge (\psi \to \phi))$$
$$(\phi \uparrow \psi) \text{ is shorthand for } (\neg(\phi \wedge \psi))$$
$$(\phi \downarrow \psi) \text{ is shorthand for } (\neg(\phi \vee \psi))$$

We are now ready to explore the utility of first-order logic in the study of mathematics, where we relate the study of first-order structures to first-order languages.

### 2.1.3 First-Order Structures Revisited

Throughout this subsection, we will fix a first-order language $\mathcal{L}$ with signature

$$\{n_i \mid i \in I\} \qquad \{m_j \mid j \in J\} \qquad K$$

where the $n_i$s are the arities of the relation symbols $R_i$, the $m_j$ are the arities of the function symbols $f_j$, and $K$ is the index set of the constants $c_k$.

**Definition 2.1.19** (First-Order Structures in First-Order Languages). A **structure** in $\mathcal{L}$, or an $\mathcal{L}$-**structure**, is a first-order structure

$$\mathcal{A} = \left\langle A; \{\overline{R_i}\}_{i\in I}, \{\overline{f_j}\}_{j\in J}, \{\overline{c_k}\}_{k\in K} \right\rangle \tag{2.1.1}$$

such that the signature of $\mathcal{A}$ is the same as that of $\mathcal{L}$, ie, the arities of the relations and functions in $\mathcal{A}$ are the same as those in $\mathcal{L}$.

If one takes a closer look at the definitions of first-order structures and first-order languages (resp. Definition 2.1.4 and Definition 2.1.9), one notices that the former involves **actual** relations and functions, which are defined as subsets of certain sets, whereas the latter merely involves function and relation **symbols**. This is a crucial distinction.

**Convention.** The reason why we put bars on top of the symbols is to distinguish functions and relations as they appear in first-order structures—*with* bars on top—from the way we express them in the ambient first-order language.

For the remainder of this subsection, fix a first-order structure $\mathcal{A}$. Denote its domain, relations, functions and constants as in (2.1.1), with bars above as per our convention for this module. We now discuss the significance of first-order structures and underscore the power and versatility of first-order languages.

When we say, in Definition 2.1.19, that the signature of an $\mathcal{L}$-structure is the same as that of the first-order language $\mathcal{L}$, we mean that the arities of the relations in the structure must match up with the arities of the relation symbols in the language, and similarly for functions and constants. More precisely, we have the following.

**Definition 2.1.20** (Interpretation). Let $\mathcal{A}$ be an $\mathcal{L}$-structure. The correspondence

$$\overline{R_i} \leftrightsquigarrow R_i \qquad \overline{f_j} \leftrightsquigarrow f_j \qquad \overline{c_k} \leftrightsquigarrow c_k$$

between relations, functions and constants in $\mathcal{A}$ and relation symbols, function symbols and constant symbols in $\mathcal{L}$ is called an **interpretation of** $\mathcal{L}$.

We have a special term for the "$\mathcal{L}$ to $\mathcal{A}$" direction of this correspondence.

**Definition 2.1.21** (Valuation). Let $\mathcal{A}$ be an $\mathcal{L}$-structure. A **valuation** in $\mathcal{A}$ is a function

$$\mathbf{v} : \{\text{Terms of } \mathcal{L}\} \to A$$

that assigns terms of $\mathcal{L}$ to their interpretations in $\mathcal{A}$ in the following manner.

1. For all constant symbols $c_k$ in $\mathcal{L}$, we have

$$\mathbf{v}(c_k) = \overline{c_k}$$

2. For all terms $t_1, \ldots, t_m$ and $m$-ary function symbols $f$ in $\mathcal{L}$, we have

$$\mathbf{v}(f(t_1, \ldots, t_m)) = \overline{f}(\mathbf{v}(t_1), \ldots, \mathbf{v}(t_m))$$

where $\overline{f}$ is the interpretation of $f$ in $\mathcal{A}$.

The idea is that a first-order language gives a purely symbolic way of expressing relationships between objects and their properties. When studying statements expressed in first-order languages, one must purely view them symbolically, as formal expressions that do not carry any *meaning* per se. The 'meaning' comes from valuations that allow us to interpret symbols in first-order languages as ideas in first-order structures.

We have an existence and uniqueness result.

**Lemma 2.1.22.** *Let $\mathcal{A}$ be an $\mathcal{L}$-structure with domain $A$. Fix elements $a_0, a_1, a_2, \ldots \in A$. There exists a unique valuation $\mathbf{v}$ of $\mathcal{L}$ in $\mathcal{A}$ such that $\mathbf{v}(x_i) = a_i$ for all $i \in \mathbb{N}$.*

*Proof.* We begin by showing existence. We can define $\mathbf{v}$ explicitly for all terms of $\mathcal{L}$ by performing recursion on their length. For terms of length 1, we deal with the variables and the distinguished constants separately.

1. $\mathbf{v}(x_i) := a_i$ for all $i \in \mathbb{N}$
2. $\mathbf{v}(c_k) := \overline{c_k}$ for all $k \in K$

We can then define $\mathbf{v}$ for terms arising from functions and relations of arity $\geq 2$ by setting

$$\mathbf{v}(f(t_1, \ldots, t_m)) := \overline{f}(\mathbf{v}(t_1), \ldots, \mathbf{v}(t_m))$$

for all terms $t_1, \ldots, t_m$ and $m$-ary function symbols $f$ in $\mathcal{L}$ with interpretation $\overline{f}$ in $\mathcal{A}$. Such a valuation is unique because if there are two valuations satisfying the condition on the variables, then they agree on all terms of length 1 (because they must agree on all constants—see Definition 2.1.21). The fact that they obey the recursion relation by definition then gives us the result. □

**Example 2.1.23** (Interpreting a Term in Groups)**.** Let $\mathcal{L}$ have the following data.

1. Function symbols: a binary symbol $m$ and a unary symbol $i$.
2. Relation symbols: a binary symbol $R$.
3. Constant symbols: a constant symbol $e$.

The following is an $\mathcal{L}$-formula:

$$m(m(x_0, x_1)\,, i(x_0))$$

Consider the first-order structure $\mathcal{A}$, meant to represent a group, with the underlying data.

1. Domain: $G$, a set.
2. Functions: the binary function $\overline{m}$ of multplication (the group operation) and the unary function $\overline{i}$ of inversion.
3. Relations: the binary relation $\overline{R}$ of equality.
4. Constants: the identity element $\overline{e}$.

Fix arbitrary group elements $g, h \in G$. We know, from Lemma 2.1.22, that there exists a unique valuation $\mathbf{v}$ of $\mathcal{L}$ in $\mathcal{A}$ such that $\mathbf{v}(x_0) = g$ and $\mathbf{v}(x_1) = h$. This valuation will allow us to interpret the formula above as a statement about the group elements $g$ and $h$:

$$\begin{aligned}
\mathbf{v}(m(m(x_0, x_1)\,, i(x_0))) &= \overline{m}(\mathbf{v}(m(x_0, x_1))\,, \mathbf{v}(i(x_0))) \\
&= \overline{m}\big(\overline{m}(\mathbf{v}(x_0)\,, \mathbf{v}(x_1))\,, \overline{i}(\mathbf{v}(x_0))\big) \\
&= \overline{m}\big(\overline{m}(g, h)\,, \overline{i}(g)\big) \\
&= \overline{m}\big(g \cdot h, g^{-1}\big) \\
&= ghg^{-1}
\end{aligned}$$

If we had defined the structure $\mathcal{A}$ a little differently—for example, if we had defined $\overline{m}(g, h) := hg$ instead—then the interpretation of the formula in $\mathcal{A}$ would have been different, despite the formula itself being exactly the same in $\mathcal{L}$. This illustrates the role of valuations and interpretations in moving between purely formal, syntactic expressions in first-order languages to meaningful expressions or statements in first-order structures.

*Remark.* It is worth remarking that in the above example, we use the $=$ symbol somewhat frivolously. On the one hand, left- and right-hand sides of each equation lie in $G$, and $=$ can be understood as equality in $\mathcal{A}$. However, one can also view these as being a *syntactic* equalities in $\mathcal{A}$, since the properties we use to manipulate the above expressions are independent of the group structure of $\mathcal{A}$. The only simplifications we make come from the *definitions* of $\mathbf{v}$, $\overline{m}$, and $\overline{i}$ (for that matter, we can even view $\cdot$ and $^{-1}$ as notation for $\overline{m}$ and $\overline{i}$ instead of viewing $\overline{m}$ and $\overline{i}$ as notation for $\cdot$ and $^{-1}$). We sidestep these syntactic nuances in this module, but we do mention

that a more syntax-heavy treatment is necessary for a computer-scientific study of first-order logic.

With this, we have studied first-order languages and structures in sufficient detail to be able to talk about how we can express and implement ideas from propositional logic in first-order logic.

## 2.2 A Bridge between Propositional and First-Order Logic

The purpose of this section is to discuss how ideas in propositional logic can be expressed in first-order logic. As we might expect from Definition 2.1.9, first-order logic is a generalisation of propositional logic. We will use this section to make this precise.

We will begin by making rigorous the notion of **logical validity**. We will see a connection with tautologies, which, as we know, are precisely the theorems of propositional logic. We will explore notions like satisfaction and substitution to make this precise.

### 2.2.1 Valuations Satisfying Formulae

For the remainder of this subection, fix some first-order language $\mathcal{L}$ and some $\mathcal{L}$-structure $\mathcal{A}$. We begin by defining a notion of equivalence for valuations that is weaker than equality.

**Definition 2.2.1** (Equivalence with respect to a Variable)**.** Suppose $x_l$ is a variable in $\mathcal{L}$ and valuations $\mathbf{v}, \mathbf{w}$ of $\mathcal{L}$ in $\mathcal{A}$. We say that $\mathbf{v}$ and $\mathbf{w}$ are $x_l$-**equivalent** if for all $i \in \mathbb{N} \setminus \{l\}$, we have $\mathbf{v}(x_i) = \mathbf{w}(x_i)$.

One way to see that this notion is weaker than equality of valuations is that if valuations agree on all variables, then by Lemma 2.1.22, they must be equal. The fact that there is one variable on which they *may* differ is what makes this notion weaker. We also emphasise that we only stipulate that $x_l$-equivalent valuations *need not* agree on $x_l$, not that they *must not* agree on $x_l$. In particular, *equal valuations are equivalent with respect to all variables*.

We are now ready to define precisely what it means for a valuation to satisfy a formula. The idea is that in a general setting—ie, in a first-order language—formulae do not express any properties, and without an interpretation in a first-order structure, it is not very meaningful to talk about what it means for a formula to be 'true'. A valuation translates formulae into 'meaningful statements', and

therefore, informally, we can think of a valuation satisfying a formula as meaning that it translates a general formula into some interpretation that we know to be 'true' in some sense.

**Definition 2.2.2** (Satisfaction). Let $\phi$ be an $\mathcal{L}$-formula and $\mathbf{v}$ a valuation of $\mathcal{L}$ in $\mathcal{A}$. We can define what it means for $\mathbf{v}$ **satisfies** $\phi$ **in** $\mathcal{A}$ recursively on the number of connectives or quantifiers in $\phi$.

1. If $\phi$ is an atomic formula, ie, has no connectives or quantifiers, then we know $\phi$ is of the form $R(t_1, \ldots, t_n)$ for some $n$-ary relation $R$ and terms $t_1, \ldots, t_n$ in $\mathcal{L}$. In this case, we say that $\mathbf{v}$ **satisfies** $\phi$ **in** $\mathcal{A}$ if the relation

$$\overline{R}(\mathbf{v}(t_1), \ldots, \mathbf{v}(t_n))$$

   holds in $\mathcal{A}$, ie, if $(\mathbf{v}(t_1), \ldots, \mathbf{v}(t_n)) \in \overline{R} \subseteq A^n$.
2. If $\phi$ is not an atomic formula in $\mathcal{L}$, then we know $\phi$ is of one of the following forms.
   (a) If $\phi$ is of the form $(\neg\psi)$ for some $\mathcal{L}$-formula $\psi$, then we say that $\mathbf{v}$ **satisfies** $\phi$ **in** $\mathcal{A}$ if $\mathbf{v}$ does not satisfy $\psi$ in $\mathcal{A}$.
   (b) If $\phi$ is of the form $(\psi \to \chi)$ for $\mathcal{L}$-formulae $\psi$ and $\chi$, then we say that $\mathbf{v}$ **satisfies** $\phi$ **in** $\mathcal{A}$ if it is not the case that $\mathbf{v}$ satisfies $\psi$ in $\mathcal{A}$ and $\mathbf{v}$ does not satisfy $\chi$ in $\mathcal{A}$.
   (c) If $\phi$ is of the form $(\forall x)\,\psi$ for some $\mathcal{L}$-formula $\psi$, we say $\mathbf{v}$ **satisfies** $\phi$ **in** $\mathcal{A}$ if for all variables $x$, if $\mathbf{w}$ is a valuation of $\mathcal{L}$ in $\mathcal{A}$ that is $x$-equivalent to $\mathbf{v}$, then $\mathbf{w}$ satisfies $\psi$ in $\mathcal{A}$.

   In all three cases, we have define what it means for $\mathbf{v}$ to satisfy $\phi$ in terms of what it means for $\mathbf{v}$ to satisfy formulae with one fewer connectives or quantifiers than $\phi$, making the recursion well-defined.

If $\mathbf{v}$ satisfies $\phi$ in $\mathcal{A}$, we write $\mathbf{v}[\phi] = \mathbf{T}$, and if $\mathbf{v}$ does not satisfy $\phi$ in $\mathcal{A}$, we write $\mathbf{v}[\phi] = \mathbf{F}$.

We remark that the notation $\mathbf{v}[\phi] = \mathbf{T}$ is merely shorthand for a statement of fact. It does not actually represent an equality in some setting where both $\mathbf{T}$ and $\mathbf{v}[\phi]$ are defined. The same goes for $\mathbf{v}[\phi] = \mathbf{F}$. However, it is often convenient to abuse notation.

**Convention.** Given valuations $\mathbf{v}, \mathbf{w}$ of $\mathcal{L}$ in $\mathcal{A}$, we write

$$\mathbf{v}[\phi] = \mathbf{w}[\phi]$$

to denote the condition that

$$\mathbf{v}[\phi] = \mathbf{T} \text{ if and only if } \mathbf{w}[\phi] = \mathbf{T}$$

or the equivalent condition that

$$\mathbf{v}[\phi] = \mathbf{F} \text{ if and only if } \mathbf{w}[\phi] = \mathbf{F}$$

This notion of a valuation satisfying a formula tells us what it means for a formula to hold true in a structure for a *given* assignment of meaning to the first-order language symbols in the formula. We can define a more absolute notion of truth.

**Definition 2.2.3** (Truth of a First-Order Formula in a First-Order Structure)**.** Let $\phi$ be an $\mathcal{L}$-formula. We say $\phi$ **is true in** $\mathcal{A}$, or that $\mathcal{A}$ **is a model of** $\phi$, if $\mathbf{v}[\phi] = \mathbf{T}$ for all valuations $\mathbf{v}$ of $\mathcal{L}$ in $\mathcal{A}$. We denote this by $\mathcal{A} \models \phi$.

A first-order formula true in one structure might not be true in another.

**Example 2.2.4.** Let $\mathcal{L}$ have one binary relation. Consider the atomic formula

$$(\forall x_1)\,(\exists x_2)\,R(x_1, x_2) \tag{2.2.1}$$

This formula is true in structures like $\langle \mathbb{N}; < \rangle$, $\langle \mathbb{Z}, < \rangle$, and $\langle \mathbb{Z}, > \rangle$.

**Non-Example 2.2.5.** Let $\mathcal{L}$ be as above. The formula (2.2.1) above is NOT true in $\langle \mathbb{N}; > \rangle$, because 0 does not have a predecessor in $\mathbb{N}$. The reason for this is that relation inputs are *ordered* (ie, relations are not, in general, symmetric).

Despite being valuation-indepentent, given that the truth of formulae is structure-dependent, it is still not the strongest notion of truth we can define in first-order logic. We want theorems to be stronger and more 'absolute' notions of truth. To that end, we define logical validity, a *structure-independent notion of truth*.

**Definition 2.2.6** (Logical Validity). Let $\phi$ be an $\mathcal{L}$-formula. We say that $\phi$ is logically valid if $\mathcal{A} \models \phi$ for all $\mathcal{L}$-structures $\mathcal{A}$.

Logically valid formulae in first-order logic are meant to be analogues of tautologies in propositional logic. An important difference, though, is that in propositional logic, there is an algorithm that *decides* whether a given formula is a tautology. In first-order logic, this is not usually the case, as we shall see when we study Gödel's Incompleteness Theorem.

**Example 2.2.7.** For all $\mathcal{L}$-formulae $\phi$, the formula

$$((\exists x_1)(\forall x_2)\,\phi \to (\forall x_2)(\exists x_1)\,\phi)$$

is logically valid.

*Proof.* `sorry` □

**Non-Example 2.2.8.** Given an $\mathcal{L}$-formulae $\phi$, the formula

$$((\forall x_1)(\exists x_2)\,\phi \to (\exists x_2)(\forall x_1)\,\phi)$$

is not necessarily logically valid.

*Proof.* `sorry` □

### 2.2.2 Substitution

In this section, we will discuss how replacing propositional variables in propositional formulae with first-order formulae gives a bridge between the two kinds of logic. For the remainder of this subection, fix

- a natural number $n$
- a first-order language $\mathcal{L}$
- $\mathcal{L}$-formulae $\phi_1, \ldots, \phi_n$
- propositional variables $p_1, \ldots, p_n$

- a propositional formula $\chi$ with propositional variables $p_1, \ldots, p_n$ (cf. Definition 1.1.3)

We have a name for replacing the $p_i$ $\chi$ with $\phi_i$.

**Definition 2.2.9** (Substitution Instance)**.** The **substitution instance of $\chi$ with $\phi_1, \ldots, \phi_n$** is the $\mathcal{L}$-formula obtained by replacing each propositional variable $p_i$ in $\chi$ with the $\mathcal{L}$-formula $\phi_i$.

*Remark.* We underscore here that a substitution instance is a formula in $\mathcal{L}$, not a propositional formula, because the terms of which it is made are all $\mathcal{L}$-formulae rather than propositional variables.

For the remainder of this subsection, denote by $\theta$ the substitution instance of $\chi$ with $\phi_1, \ldots, \phi_n$.

Substitution is our gateway for 'embedding' propositional logic into first-order logic. This 'embedding' preserves 'truth' in the following manner.

**Theorem 2.2.10.** *If $\chi$ is a tautology, then $\theta$ is logically valid.*

*Proof.* Assume that $\chi$ is a tautology (cf. Definition 1.1.9). To show that $\theta$ is logically valid, we need to show that $\mathcal{A} \models \theta$ for all $\mathcal{L}$-structures $\mathcal{A}$. To that end, let $\mathcal{A}$ be an $\mathcal{L}$-structure and let $\mathbf{v}$ be a valuation of $\mathcal{L}$ in $\mathcal{A}$. We need to show that $\mathbf{v}[\theta] = \mathbf{T}$. `sorry` □

It is tempting to ask whether all logically valid formulae are substitution instances of tautologies. Such a fact would truly make the above result a two-way bridge. Unfortunately, this is not true, as shown by the following counterexample.

**Counterexample 2.2.11.** For all $\mathcal{L}$-formulae $\phi$, the $\mathcal{L}$-formula

$$((\exists x_2)\,(\forall x_1)\,\phi \to (\forall x_1)\,(\exists x_2)\,\phi)$$

is logically valid. However, it is not a substitution instance of any propositional formula, because there is no notion of quantification in propositional logic.

This tells us that first-order logic is not only stronger than propositional logic, it is **strictly** stronger.

## 2.3 Variables and the Universal Quantifier

The purpose of this section is to understand how to work with variables and the universal quantifier in first-order languages. Throughout this section, fix a first-order language $\mathcal{L}$.

### 2.3.1 Bound and Free Variables

Consider the formula

$$\psi_1 : (R_1(x_1, x_2) \to (\forall x_3)\, R_2(x_1, x_3)) \tag{2.3.1}$$

where $R_1, R_2$ are relation symbols in $\mathcal{L}$ and $x_1, x_2, x_3$ are variables in $\mathcal{L}$. Intuitively, we do not want $x_3$ to 'exist', or be 'known' or 'accessible', outside of the sub-formula[1] $(\forall x_3)\, R_2(x_1, x_3)$. We can mae this notion precise.

**Definition 2.3.1** (Scope of a Quantifier)**.** Suppose $\phi$ and $\psi$ are $\mathcal{L}$-formulae. If $(\forall x_i)\, \phi$ occurs as a sub-formula of $\psi$, we say that $\phi$ is the **scope of** $(\forall x_i)$ **in** $\psi$.

**Example 2.3.2.** In (2.3.1), $R_2(x_1, x_3)$ is the scope of the quantifier $(\forall x_3)$.

We use the existence of a scope to characterise variables.

**Definition 2.3.3** (Bound Variables)**.** Let $\psi$ be an $\mathcal{L}$-formula and let $x_j$ be a variable that appears in $\psi$. We say that $x_j$ is **bound in** $\psi$ if it is within the scope of a quantifier.

**Example 2.3.4.** In (2.3.1), the variable $x_3$ is bound, because it lies in the scope $R_2(x_1, x_3)$ of the quantifier $(\forall x_3)$.

We also have a special name for variables that are not bound.

[1] When we say sub-formula, we mean exactly what it sounds like: since formulae are built from smaller formulae using connectives and the quantifier, when we say sub-formula, we mean a formula that arose at an intermediate step in this inductive construction of the formula of which it is a sub-formula.

**Definition 2.3.5** (Free Variables). Let $\psi$ be an $\mathcal{L}$-formula and let $x_j$ be a variable that appears in $\psi$. We say that $x_j$ is **free in** $\psi$ if it is not bound in $\psi$.

**Example 2.3.6.** In (2.3.1), the variable $x_1$ is bound, because it does not lie in the scope of any quantifiers, meaning it is not bound.

The same variable can have both free and bound occurrences within a given formula.

**Example 2.3.7.** In the formula

$$((\forall x_1)\, R_1(x_1, x_2) \to R_2(x_1, x_2))$$

the variable $x_1$ is bound in the expression

$$(\forall x_1)\, \underbrace{R_1(x_1, x_2)}_{\text{scope of } (\forall x_1)}$$

but free in the expression

$$R_2(x_1, x_2)$$

We can also show that existentially quantified variables are bound.

**Lemma 2.3.8.** *Let $\psi$ and $\phi$ be an $\mathcal{L}$-formulae. If $\psi$ contains*

$$(\exists x_1)\, \phi$$

*as a sub-formula, then $x_1$ is bound in $\psi$.*

*Proof.* This follows immediately from the definition of the existential quantifier. `sorry` □

The occurrence of free variables inside a formula means that it is very general, as the variables in it are purely formal symbols that we have no machinery to reason with. We give a special term to, and have a special interest in, formulae with no free variables.

**Definition 2.3.9** (Closed Formulae)**.** An $\mathcal{L}$-formula with no free variables is called a **closed formula** or a **sentence**.

Formulae that are not closed can be thought of as being 'dependent' on their free variables, in the sense that we would intuitively want to only define a notion of substitution only for free variables. To that end, we adopt some notation.

**Convention.** Let $\psi$ be an $\mathcal{L}$-formula. If $\mathcal{L}$ has free variables $x_1, \ldots, x_n$, then we denote $\psi$ by

$$\psi(x_1, \ldots, x_n)$$

when we wish to underscore the dependence of $\psi$ on $x_1, \ldots, x_n$ or the fact that $x_1, \ldots x_n$ are free in $\psi$ or perform a substitution.

We now define substitution.

**Definition 2.3.10** (Substitution)**.** Let $t_1, \ldots, t_n$ be terms in $\mathcal{L}$ and $\psi(x_1, \ldots, x_n)$ an $\mathcal{L}$-formula with free variables $x_1, \ldots, x_n$[a]. We define the formula obtained by **substituting the $t_i$ for the $x_i$** to be the $\mathcal{L}$-formula obtained by replacing each occurrence of $x_i$ in $\psi$ with an occurrence of the corresponding $t_i$. We denote this formula

$$\psi(t_1, \ldots, t_n)$$

[a] By the aforementioned convention, we would ordinarily not mention that $x_1, \ldots, x_n$ are free in $\psi$.

## 2.3.2 An Analogue of Completeness

Throughout this subsection, fix an $\mathcal{L}$-formula $\phi$ and an $\mathcal{L}$-structure $\mathcal{A}$.

We begin by mentioning a result about valuations.

**Theorem 2.3.11.** *Fix $n \in \mathbb{N} \cup \{0\}$. If $\phi$ has free variables $x_1, \ldots, x_n$ and $\mathbf{v}, \mathbf{w}$ are valuations*

*of $\mathcal{L}$ in $\mathcal{A}$ with $\mathbf{v}(x_i) = \mathbf{w}(x_i)$ for all $1 \leq i \leq n$, then*

$$\mathbf{v}[\phi] = \mathbf{T} \text{ if and only if } \mathbf{w}[\phi] = \mathbf{T}$$

*In other words, we have $\mathbf{v}[\phi] = \mathbf{w}[\phi]$.*

*Proof.* Fix valuations $\mathbf{v}$ and $\mathbf{w}$ of $\mathcal{L}$ in $\mathcal{A}$ that agree on all the free variables of $\phi$. We argue that $\mathbf{v}[\phi] = \mathbf{w}[\phi]$ by performing induction on the total number connectives and quantifiers in $\phi$.

The base case is when $\phi$ has no connectives or quantifiers, ie, when $\phi$ is atomic in terms that contain no quantifiers. In this case, we have that $\phi$ is of the form

$$R(t_1, \ldots, t_m)$$

where $t_1, \ldots, t_m$ are terms that contain no quantifiers. In this case, each $t_i$ is either a constant or a variable. Since $\phi$ is assumed not to contain any quantifiers, the $t_i$ that are variables must be free variables, and the rest must be constants. Since $\mathbf{v}$ and $\mathbf{w}$ agree on all constants by definition and on all free variables by assumption, we must have $\mathbf{v}(t_i) = \mathbf{w}(t_i)$ for all $i$. Then, `sorry` □

This gives us a completeness-like result about closed formulae.

**Corollary 2.3.12.** *If $\phi$ is closed, then either $\mathcal{A} \models \phi$ or $\mathcal{A} \models (\neg\phi)$.*

*Proof.* If $\phi$ is closed, $\phi$ has no free variables. Then, any two valuations of $\mathcal{L}$ in $\mathcal{A}$ agree vacuously on the set of all free variables of $\phi$. Therefore, there cannot be two distinct valuations such that one satisfies $\phi$ and one does not. In other words, either $\phi$ is true in $\mathcal{A}$ or $\phi$ is not true in $\mathcal{A}$. Equivalently, either $\phi$ or $(\neg\phi)$ is true in $\mathcal{A}$. □

### 2.3.3 Understanding the Universal Quantifier

The purpose of this section is to discuss the relationship between the syntax and the semantics of the universal quantifier $\forall$. We have not been too precise so far about what it *means* to write an expression like

$$(\forall x_1)\, \phi$$

for some formula $\phi$ with a bound variable $x_1$. We particularly delve into nuances that arise due to the fact that in our strict syntactic definition of $\mathcal{L}$-formulae (Definition 2.1.15, we only allow the quantifier to be succeeded by a **variable** in $\mathcal{L}$, which must be one of $x_1, x_2, \ldots$. The really confusing thing about this syntactic definition of the universal quantifier is the fact that it is not obvious that this quantifier means "for all": instead of saying "for all variables $x_1$, we have $\phi$", on a strictly syntactic level, the expression $(\forall x_1)\,\phi$ is using the *specific variable* $x_1$.

What we would want—say, in an interactive theorem prover like Lean—is a mechanism to be able to *remove* the quantifier and instead *introduce* a free variable into our formula and label that variable differently from all other free and bound variables in our formula.

# Exercises

Here, we provide solutions to the exercises discussed in problems classes. We do not include *all* exercises, only the ones that are 'interesting'.

## Problems Class 1

**Exercise 1.1.** *Decide whether the following are true or false, giving reasons. Here,* $\Gamma$ *is a set of* **L***-formulae, as are* $\phi$ *and* $\psi$*.*

1. *In every* **L***-formula, the number of opening parentheses* ( *is equal to the number of connectives.*
2. *If* $\Gamma \vdash_{\mathbf{L}} \phi$ *and* $\Gamma \vdash_{\mathbf{L}} (\phi \to \psi)$*, then* $\Gamma \vdash_{\mathbf{L}} \psi$*.*
3. *Suppose* $\mathbf{v}$ *is a propositional valuation and* $\Gamma$ *is such that* $\mathbf{v}(\Gamma) = \mathbf{F}$*. Then, for all* $\phi$ *such that* $\Gamma \vdash_{\mathbf{L}} \phi$*, we have* $\mathbf{v}(\phi) = \mathbf{F}$*.*
4. *Suppose* $\mathbf{v}$ *is a propositional valuation and* $\Delta_{\mathbf{v}} = \{\phi \mid \mathbf{v}(\phi) = \mathbf{F}\}$*. Then,* $\Delta$ *is consistent.*
5. *Suppose* $\mathbf{v}$ *is a propositional valuation and* $\Delta_{\mathbf{v}} = \{\phi \mid \mathbf{v}(\phi) = \mathbf{F}\}$*. Then,* $\Delta$ *is complete.*

*Solution.*

1. TRUE. Every connective in an **L**-formula is associated with precisely one sub-**L**-formula that is not a variable, and the number of pairs of parentheses (which is the number of opening parentheses) is precisely the number of sub-**L**-formulae that are not variables, because every such sub-**L**-formula is enclosed in parentheses.

2. TRUE. This is simply modus ponens generalised to arbitrary (potentially nonempty) $\Gamma$. While the deduction rule (MP) is technically not something we defined over arbitrary contexts $\Gamma$,

we can easily prove that it holds by induction on the length of $\Gamma$.

3. FALSE. Let $\mathbf{v}$ be any propositional valuation and $\phi$ and axiom of $\mathbf{L}$.

4. FALSE. Let $\mathbf{v}$ be any propositional valuation and $p$ a propositional variable. Then, we know $(p \to (\neg p)) \in \Delta_{\mathbf{v}}$. We also know that $(p \wedge (\neg p)) \in \Delta_{\mathbf{v}}$. But the latter is logically equivalent to the negation of the former, making $\Delta_{\mathbf{v}}$ inconsistent.[2]

5. TRUE. Let $\mathbf{v}$ be `sorry`

**Exercise 1.2.** *Suppose $\phi$ is an $\mathbf{L}$-formula and $\Gamma$ is a set of $\mathbf{L}$-formulae. Do the following **syntactically**, ie, without using the Completeness Theorem. You may use theorems of $\mathbf{L}$ we have proved in lectures or the weekly problem sheets.*

(i) *Express the 'Law of the Excluded Middle' $(\phi \vee (\neg\phi))$ as an $\mathbf{L}$-formula and say why this is a theorem of $\mathbf{L}$.*

(ii) *`sorry`*

*Solution.*

(i) Recall that any formula of the form $(a \vee b)$ is expressed as $((\neg a) \to b)$. Thus, we have that $(\phi \vee (\neg\phi))$ is expressible as

$$((\neg\phi) \to (\neg\phi))$$

Indeed, we proved in lectures that in $\mathbf{L}$, any formula implies itself. Therefore, the Law of the Excluded Middle is a theorem of $\mathbf{L}$.

(ii) `sorry`

**Exercise 1.3.** *Show that the set of connectives $\{\neg, \leftrightarrow\}$ is not adequate.*

*Hint.* See EdStem.

*Solution.* The idea is to lok at all possible truth functions of two variables $p$ and $q$ that are obtained using formulae involving $\neg$ and $\leftrightarrow$. `sorry`

[2] We have actually done something stronger than simply provide a counterexample: we have proved that the statement is false for *any* propositional valuation, not just for *some* propositional valuation

# Problems Class 2

**Exercise 2.4.** *In Example 2.1.23, we introduced a first-order language $\mathcal{L}$ that is appropriate for groups. In this question, we will use the notation defined in Example 2.1.23. Define the $\mathcal{L}$-structures*

$$\mathcal{A} := \langle \mathbb{Z}; =, +, -, 0 \rangle$$
$$\mathcal{B} := \langle \mathbb{Q}; =, +, -, 0 \rangle$$

1. *True or false? Give reasons.*
   (a) *Every $\mathcal{L}$-structure is a group.*
   (b) $(\neg m(x_2, m(e, x_1)))$ *is a term of* $\mathcal{L}$.
   (c) $(\neg m(x_2, m(e, x_1)))$ *is a formula of* $\mathcal{L}$.
   (d) $R(e, m(x_1, i(x_1)))$ *is a formula of* $\mathcal{L}$.
   (e) $(\exists x_1)(\neg R(e, m(x_1, i(x_1))))$ *is a formula of* $\mathcal{L}$.
   (f) $(\exists x_1)((\neg R(e, m(x_1, i(x_1)))))$ *is a formula of* $\mathcal{L}$.
2. *Suppose* $\mathbf{v}$ *is a valuation of* $\mathcal{L}$ *in* $\mathcal{A}$ *such that*

$$\mathbf{v}(x_1) = 2$$
$$\mathbf{v}(x_2) = 4$$
$$\mathbf{v}(x_j) = 0 \text{ for } j \neq 1, 2$$

   *(a) Compute*

$$\mathbf{v}(m(x_2, m(e, x_1)))$$

   *(b) Find a term $t$ of $\mathcal{L}$ such that $\mathbf{v}(t) = -6$.*
   *(c) Is there a term $t'$ of $\mathcal{L}$ such that $\mathbf{v}(t') = 7$?*
3. *Find a closed $\mathcal{L}$-formula $\phi$ such that $\mathcal{A} \models \phi$ but $\mathcal{B} \not\models \phi$.*

*Solution.*

1. (a) FALSE. The first-order language $\mathcal{L}$ is syntactic. There is nothing that tells us that the unary function symbol for inversion must actually correspond to a function that undoes

the binary function to get the constant. For example,

$$\langle \mathbb{Z}; \times, -, 0 \rangle$$

is, as per Definition 2.1.19, an $\mathcal{L}$-structure, because it has the same signature as $\mathcal{L}$.

(b) FALSE. As per Definition 2.1.11, a term cannot have a connective. Since the expression

$$(\neg m(x_2, m(e, x_1)))$$

has a connective, it cannot be a term.

(c) FALSE. As per Definition 2.1.15, any formula must contain an atomic formula, which contains a relation symbol. Since the expression

$$(\neg m(x_2, m(e, x_1)))$$

has no relation symbols, it cannot be a formula.

(d) TRUE. Both $e$ and $m(x_1, i(x_1))$ are terms, as they satisfy Definition 2.1.11. Then, since $R$ is a relation symbol, we have that the expression

$$R(e, m(x_1, i(x_1)))$$

is, in fact, an *atomic* formula (cf. Definition 2.1.14). Therefore, by Definition 2.1.15, it is also a formula.

(e) TRUE. This is easily checked.

(f) FALSE. It looks the same as the previous one, but there are two brackets too many! So, the formula is **not well-formed**. (In practice, we only use brackets to disambiguate. In this module, though, we've defined them as an integral part of our syntax. Therefore, in this module, we have to be careful when dealing with brackets!)

2. (a) In $\mathcal{A}$, the (additive) group of integers, this corresponds to the expression

$$4 + 2$$

which we can evaluate to 6.

(b) The following is such a term.

$$m(i(2)\,,m(i(2)\,,i(2)))$$

It corresponds to the expression

$$(-2)+((-2)+(-2))$$

which we can simplify, using the properties of $\mathcal{A}$, to $-6$.

(c) We cannot, because as per our definition of $\mathbf{v}$, the valuation of any term must be an even number.

For the latest version of these notes, visit https://thefundamentaltheor3m.github.io/LogicNotes/main.pdf. For any suggestions or corrections, please feel free to fork and make a pull request to my repository.