Part I: Propositional Logic

Contents
§1. Introduction
§2. Formalization
§3. Propositional Syntax
§4. Propositional Semantics
§5. Basic Logical Concepts
§6. Introducing Semantic Tableaux
§7. Tableau Methods I
§8. Tableau Methods II
§9. Tableau Methods III
§10. Tableau Methods IV
§11. Equivalence, Truth Functions and Adequacy
§12. Soundness, Completeness and Decidability; Other Deductive Systems
§1. Introduction

1.1 What is an Argument?

Logic is concerned with the validity of arguments. So, what is an argument? You may (or may not) recall the famous “Argument Sketch” from Monty Python:

Man: Oh look, this isn’t an argument.
Arguer (John Cleese): Yes it is.
M: No, it isn’t. It's just contradiction.
A: No, it isn't.
M: It is!
A: It is not.
M: Look, you just contradicted me.
A: I did not.
M: Oh, you did!!
A: No, no, no.
M: You did just then.
A: Nonsense!
M: Oh, this is futile!
A: No, it isn't.
M: I came here for a good argument.
A: No, you didn’t; no, you came here for an argument.
M: An argument isn’t just contradiction.
A: It can be.
M: No, it can't. An argument is a connected series of statements intended to establish a proposition.
A: No, it isn't.
M: Yes it is! It’s not just contradiction.
A: Look, if I argue with you, I must take up a contrary position.
M: Yes, but that’s not just saying ‘No, it isn’t.’
A: Yes, it is!
M: No, it isn’t!
A: Yes, it is!
M: Argument is an intellectual process. Contradiction is just the automatic gainsaying of any statement the other person makes.
(short pause)
A: No, it isn't.
1.2 Logic and Argumentation

Logic can be thought of as the science of arguments, and in particular is concerned with a key property that some arguments possess: **validity**.

*Some arguments are valid and some are invalid.*

Why is this? This is the topic of logic.

Here are some examples of arguments:

1. If it is snowing, then it is cold outside.
   
   It is snowing
   
   *Therefore*, it is cold outside.

2. If it is snowing, then it is cold outside.
   
   It is cold outside
   
   *Therefore*, it is snowing.

3. Either you are a Liverpool fan or a Manchester United fan.
   
   You are not a Manchester United fan
   
   *Therefore*, you are a Liverpool fan.

4. All humans are green.
   
   Some green things are edible
   
   *Therefore*, some humans are edible.

5. All humans are green.
   
   Some green things are not edible
   
   *Therefore*, some humans are not edible.

6. All glorphs are wibbles
   
   Some wibbles are fnoffles.
   
   *Therefore*, some glorphs are fnoffles.

Some of these arguments are **valid**, and some are not **valid**. You may at the moment only have an **intuitive** understanding of which is which. At the end of this course, you should be able to **correctly classify** these arguments as either valid or invalid, and you should be able to **explain why**.

So, one of the tasks of logic is provide a **precise mathematical theory of validity of arguments**.
1.3 Logic and Philosophy

The discipline of Logic is closely associated with Philosophy. The main branches of philosophy are:

1. **Metaphysics**: the theory of existence and being;
2. **Epistemology**: the theory of knowledge, evidence and justification;
3. **Logic**: the theory of validity;
4. **Ethics/Aesthetics**: the theory of values.

According to traditional lore, philosophical investigation of the problems of existence, knowledge, reasoning and values began with the Ancient Greeks. Especially important are Plato and Aristotle of fourth century BC Athens. Although Plato was interested in logic, it was his pupil Aristotle who developed the earliest logical theory, called the syllogism.

Aristotle’s syllogistic logical theory remained largely unchanged for over two thousand years, until the middle of the nineteenth century. A few philosophers, such as René Descartes and Immanuel Kant, thought that Aristotelian logic was in some sense complete or finished! Other philosophers, like Gottfried Leibniz, tried valiantly to introduce new ideas into logic, but these efforts led to failure, mainly because they failed to discover the central idea of quantificational variables and quantifiers.

The situation changed in a dramatic way between roughly 1850 and 1885, when logic was transformed by three mathematicians:

- George Boole (England),
- Gottlob Frege (Germany)
- Charles Peirce (USA).

The logic we learn today is called formal logic or, better, mathematical logic. Logic has become an important branch of mathematics. Some would say that logic provides the foundation of mathematics. Also, modern logic has many applications outside philosophy: for example, in mathematics, computer science and psychology.

1.4 What is Logic?

One standard characterization of logic is this:

**LOGIC** is the science or study of valid arguments.

An argument, in the sense understood by a logician (as opposed to the sense which simply means a “disagreement”: recall the Monty Python sketch) is an attempt to establish a conclusion.

An argument consists of some premises and a conclusion.

Given an argument, we want to know if the argument is valid. Does the conclusion follow logically from the premises? Is the conclusion a logical consequence of the premises? To provide a clear analysis of the logical concepts of “validity” and “logical consequence”, and to provide methods for classifying arguments as valid or invalid, is the central task of logic.

It turns out that the core logical concepts (validity, logical consequence, consistency) are intimately related to the notion of truth. For example (we shall study these definitions more carefully later),
A set of sentences is CONSISTENT if it is possible for them all to be simultaneously true.

An argument is VALID if it is impossible for the premises to be true, with the conclusion false.

A statement is a LOGICAL CONSEQUENCE of some assumptions or premises just in case, if the assumptions are true of any situation, then so must be the conclusion.

This intimate relation between logic and the notion of truth is what led Frege to regard logic as the science of truth.

1.5 Argument Structure

Intuitively and informally, an argument is a piece of reasoning, expressed by a sequence of sentences in some natural language: an argument is “a connected series of statements intended to establish a proposition”. Making this a little more precise, any argument can be put into the following form:

**Standard Argument Form:**

Premise 1

Premise 2

...

Premise n

Therefore, Conclusion

A premise is an assumption. To stress, this assumption is made “for the sake of argument”.

In writing down a premise, we do NOT assert that it is true.

In studying the validity of arguments, we are investigating what would logically follow IF it were true.

Note that an argument may be valid even if

- the premises are false and the conclusion is true
- the premises are false and the conclusion is false
- the premises are true and the conclusion is true.

However, an argument cannot be valid if the premises are true and the conclusion is false.

Thus, in logic we are interested in whether the conclusion is a logical consequence of the premises. If the conclusion does logically follow, then the argument is valid. If the conclusion does not logically follow, then the argument is invalid.
For example, here are some arguments, in Standard Argument Form:

(1) Either Manchester will win or Liverpool will win  [Premise 1]
Manchester will not win  [Premise 2]
Therefore, Liverpool will win.  [Conclusion]

(2) If that’s justice, then I’m a banana
I am not a banana
Therefore, That is not justice.

(3) John is happy only if Yoko is away
John is happy
Therefore, Yoko is away

Each of the above arguments is, in fact, valid.

Here is an invalid argument,

(4) If it has rained, then the grass is wet
The grass is wet
Therefore, It has rained.

This is a famous fallacy. The Fallacy of Affirming the Consequent. Intuitively at this stage, it can be seen that the conclusion is not a logical consequence of the premises. Later in the course, you should be able to explain why, and even prove that this is the case.

§2. Formalization

2.1 Introduction

In logic, the analysis of informal arguments proceeds by a technique known as formalization. This involves two steps:

(a) translate informal statements into a symbolic or formal language;
(b) apply mathematical methods to the resulting formalized arguments.

This may be illustrated by an example.

Consider the argument,

(1) Either Manchester will win or Liverpool will win  [Premise 1]
Manchester will not win  [Premise 2]
Therefore, Liverpool will win.  [Conclusion]
The first thing we do is to replace the component sub-sentence by symbols standing for sentences. We shall use the letters ‘P’ and ‘Q’.

Thus, (1) becomes,

(2)  Either P or Q [Premise 1]

not:-P               [Premise 2]

Therefore,  Q [Conclusion]

We have simply replaced the sub-sentences by letters: ‘Manchester will win’ by the letter ‘P’. ‘Liverpool will win’ by the letter ‘Q’.

These letters, ‘P’ and ‘Q’, are called sentence letters or statement letters.

A note on terminology: some authors call the letters P, Q, etc., propositional constants or propositional atoms. Some also call them propositional variables, but this latter expression is bad terminology, since, as we will see in the section on semantics for propositional logic, their interpretation does not vary within a particular truth value assignment, and hence they do not play the logical role of variables. Instead, they are constants relative to a particular assignment.

Examining (2), we see that there are certain “connecting” expressions remaining:

Either … or
Not.

Such connecting expressions play a central role in logic.

They are called logical connectives.

The next step is to symbolize these logical connectives. The usual symbols are

\[ \lor \] for ‘either … or …’
\[ \neg \] for ‘not’

Using these symbols, we get:

(3)  P \lor Q

\neg P

Therefore,  Q

Notice that we now have things like ‘P \lor Q’ and ‘\neg P’.

These are called compound or molecular formulas, while the sentence letters ‘P’ and ‘Q’ from which they are built are called atomic formulas.
A note on terminology: within the context of propositional logic, the terms ‘sentence’ and ‘formula’ will be used interchangeably. However, ‘formula’ is technically the more general of the two terms, and in the context of predicate logic, the sentences will comprise only a **proper subset** of the set of formulas.

Roughly, a formula is a way of expressing the “abstract logical structure” or “logical form” of a sentence.

(3) is our first example of a **formal argument**. Comparing the informal and the formal arguments, we get:

<table>
<thead>
<tr>
<th>Informal Argument</th>
<th>Formal Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>either Manchester will win or Liverpool will win</td>
<td>P ∨ Q</td>
</tr>
<tr>
<td>Manchester will not win</td>
<td>¬P</td>
</tr>
<tr>
<td>Therefore, Liverpool will win.</td>
<td>Therefore, Q</td>
</tr>
</tbody>
</table>

### 2.2 Sequents

Formal arguments crop up so often in logic that we prefer a simpler and more convenient way of writing them down. Instead of the above representation of a formal argument,

\[
P \lor Q, \neg P \quad : \quad Q
\]

we write instead:

\[
P \lor Q, \neg P \quad : \quad Q
\]

The expression above is called a sequent.

**Definition:** A sequent is a sequence of formulas of the following form,

\[
A_1, A_2, \ldots, A_n : B
\]

where \( A_1, A_2, \ldots, A_n \) and \( B \) may be any formulas.

The formulas \( A_1, \ldots, A_n \) are called the **premises** of the sequent.

The formula \( B \) is its **conclusion**.

(We shall also allow the “null case” where there are **no** premises at all.)

For example, in the above sequent,

- \( A_1 \) is the formula \( P \lor Q \);
- \( A_2 \) is the formula \( \neg P \);
- \( B \) is the formula \( Q \).

In logic, we want to devise techniques to classify a given sequent as either **valid** or **invalid**. But what does “valid” mean? In the next few lectures, you will learn exactly what “valid” means and you will learn some methods for checking if a sequent (i.e., a formal argument) is indeed valid or not. Roughly speaking, a valid sequent is one
whose conclusion is a logical consequence of the premises. Otherwise, the argument is not valid.

2.3 Propositional Connectives

Certain English expressions like,

- and,
- or,
- not,
- if \( \ldots \) then,
- if and only if

are called **propositional connectives** (or ‘logical’ or **truth-functional connectives**). They operate on sentences of English to form new sentences.

The rules (of **grammar** or **syntax**) governing these connectives are quite simple:

(i) If \( A \) is a sentence, then ‘not \( A \)’ is a sentence.

(ii) If \( A \) and \( B \) are sentences, then ‘\( A \) and \( B \)’ and ‘\( A \) or \( B \)’ and ‘if \( A \) then \( B \)’ and ‘\( A \) if and only if \( B \)’ are all sentences.

The logical connectives are standardly **symbolized** in formal logic as follows:

<table>
<thead>
<tr>
<th>English Expression</th>
<th>Logical Symbolization</th>
<th>Name of Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>not ( A )</td>
<td>( \neg A )</td>
<td>negation</td>
</tr>
<tr>
<td>( A ) and ( B )</td>
<td>( A \land B )</td>
<td>conjunction</td>
</tr>
<tr>
<td>( A ) or ( B )</td>
<td>( A \lor B )</td>
<td>Disjunction</td>
</tr>
<tr>
<td>if ( A ) then ( B )</td>
<td>( A \rightarrow B )</td>
<td>conditional</td>
</tr>
<tr>
<td>( A ) if and only if ( B )</td>
<td>( A \leftrightarrow B )</td>
<td>biconditional</td>
</tr>
</tbody>
</table>

The **negation connective** \( \neg \) is called a 1-place (or **unary**) connective, because it operates on just one sentence to form a new one. The other connectives \( \land, \lor, \rightarrow \) and \( \leftrightarrow \) are called 2-place (or **binary**) connectives, because they join a pair of sentences.

2.4 Some Terminology

Now that we have introduced the main five logical connectives, we can introduce some terminology use for discussing formulas containing these connectives:

- A **conjunction** is a formula of the form \( A \land B \); we say that \( A \) and \( B \) are its **conjuncts**.
- A **disjunction** is a formula of the form \( A \lor B \), we say that \( A \) and \( B \) are its **disjuncts**.
- A **conditional** is a formula of the form \( A \rightarrow B \), we say that the formula \( A \) is the **antecedent** and that \( B \) is **consequent**.
2.5 Examples of Formalization

We state beforehand what sentence letters symbolize what statements. E.g.,

\[ \begin{align*}
P & : \text{John is happy} \\
Q & : \text{Yoko is sad} \\
R & : \text{Paul will sing}
\end{align*} \]

Then, we obtain the following formalizations:

\[ \begin{align*}
(1) & \quad \text{John is not happy} \quad \neg P \\
(2) & \quad \text{If John is happy then Yoko is sad} \quad P \rightarrow Q \\
(3) & \quad \text{John is happy and Yoko is sad} \quad P \land Q \\
(4) & \quad \text{Either John is happy or Yoko is sad} \quad P \lor Q \\
(5) & \quad \text{John is happy if and only if Yoko is sad} \quad P \leftrightarrow Q
\end{align*} \]

2.6 Compound Formulas

In propositional logic, we start with the following ingredients:

(a) sentence letters: \(P, Q, R, \ldots\)

(b) logical connectives: \(\neg, \land, \lor, \rightarrow, \leftrightarrow\)

An amalgam of symbols obtained correctly from these ingredients is called a compound or molecular formula. For example, the following expressions are compound formulas:

\[ \begin{align*}
P \rightarrow P \\
(P \land Q) \rightarrow R \\
(P \lor Q) \rightarrow (Q \land P)
\end{align*} \]

and so on.

Notice that parentheses or brackets are used to indicate grouping.

So, the formula

\[(P \land Q) \rightarrow R\]

is quite distinct from the formula

\[P \land (Q \rightarrow R).\]

This is analogous to the use of brackets in algebraic expressions, where \((x + y)z\) is quite different from \(x + (yz)\). For example, \((3 + 5) \times 4\) is not the same as \(3 + (5 \times 4)\).

2.7 Sentence Letters and Meta-Variables

So far, we have used two different types of symbols when talking about formulas, and the distinction is an important one. In these notes, we adopt the convention that bold face upper case letters ‘A’, ‘B’ and ‘C’, etc. (with or without subscripts) stand for arbitrary formulas.

In contrast, the upper case letters ‘P’, ‘Q’, ‘R’, etc., (with or without numerical subscripts) are treated as sentence letters (again, some authors call them “propositional constants or atoms”). So the letters ‘P’, ‘Q’, ‘R’ are particular symbols in a logical language, often called the ‘object language’, since it is the object of our formal study.
The letters ‘A’, ‘B’, ‘C’ play a different role. They are not sentence letters. They are elements of the **metalanguage** that we use when talking about formulas of the object language. So they are variables of our meta-language, used to make statements of generality, and hence are called “**meta-variables**”. Roughly speaking, when we say “Let \( A \) be a formula …”, then this is a way of **referring generally to any arbitrary formula**, such as \( \neg P, P \lor Q, (P \land Q) \rightarrow P \), etc. This is comparable to the use of variables like \( x \) and \( y \) in algebra to make general statements about numbers.

Some authors of logical texts (e.g., Hodges) use **Greek** letters ‘\( \varphi \)’, ‘\( \psi \)’, etc. as meta-variables for formulas. For example, in Hodges, you might find statements like

\[
\text{A conjunction formula } \varphi \land \psi \text{ implies } \psi;
\]

In our course, we shall write instead,

\[
\text{A conjunction formula } A \land B \text{ implies } A
\]

Note that this is a **general** statement and holds for **any** conjunction formula whatever, e.g. \( P \land Q \) implies \( P \), \( Q \land R \) implies \( Q \), \( (Q \lor R) \land (P \rightarrow Q) \) implies \( Q \lor R \), etc. This is analogous to the general arithmetic statement that \( x + y = y + x \), for any numbers \( x \) and \( y \).

---

**A note on use versus mention**: most of the time, language is used to talk about non-linguistic entities and states of affairs, such as dogs, cats and football matches. However, sometimes languages is not used but rather mentioned, as in the observation that ‘cat’ is a 3 letter word. In the context of these lecture notes, when we talk about particular sentence letters such as ‘\( P \)’ and ‘\( Q \)’, these symbols are normally being **mentioned**, and the standard convention in English is to use **quotation marks** to indicate this. However, it is notationally inconvenient to constantly refer to these letters using explicit quotation marks, and you may have noticed that in many preceding cases we have omitted them. In such contexts, where it appears that the object language expression is being **mentioned** and hence quotation marks are technically called for, we adopt the liberating view that the object language symbols are **used in the metalanguage to name themselves**. This **autonomous** interpretation of object language symbols eliminates the need to worry about overly fussy quotation conventions when doing logic.

---

**2.8 Other Propositional Connectives in English**

Several other words in English function as logical connectives. Examples are:

- **unless**,
- **but**,
- **only if**,
- **just in case**,

It turns out that each of these can be represented by the connectives we have already encountered. Here are the correspondences:
### English connective | Corresponding Phrase | Formalization
--- | --- | ---
Unless A, B | A or B | A ∨ B
A, but B | A and B | A ∧ B
A only if B | If A then B | A → B
A just in case B | A if and only if B | A ↔ B

For example,

1. **Unless** John is happy, Yoko is sad
   
   P ∨ Q

2. John is happy **but** Yoko is sad
   
   P ∧ Q

3. John is happy **only if** Yoko is sad
   
   P → Q

4. John is happy **just in case** Yoko is sad
   
   P ↔ Q

Notice that the ‘if … then’ construction in English can be reversed. E.g.,

5. Yoko is sad **if** John is happy
   
   P → Q

This means exactly the same as

6. **If** John is happy, **then** Yoko is sad

However, (5) does not mean the same as,

7. Yoko is sad **only if** John is happy

So you must distinguish sharply between the connectives ‘if’ and ‘only if’.

### 2.9 Formalizing Compound Sentences

It is now possible to formalize complex English sentences, by formalizing each appearance of the logical connectives. For example, first identify the connectives:

1. **If** John is happy **and** Yoko is sad, **then** Paul will **not** sing

   and put the component sub-sentences in appropriate brackets

2. **If** [(John is happy) **and** (Yoko is sad)], **then** [Paul will **not** sing].

Now formalize in a natural way:

3. (P ∧ Q) → ¬R

Notice that you must include brackets here to indicate the grouping of sentence letters.

Similarly,

4. **Unless** John is happy **or** Paul will **not** sing, Yoko is **not** sad

This is naturally formalized as:

5. (P ∨ ¬R) ∨ ¬Q

### 2.10 Notational Variations

Note that some logical authors use different symbols for the connectives.
Here is a summary:

<table>
<thead>
<tr>
<th>Connective</th>
<th>Our Symbolization</th>
<th>Common Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>$\neg A$</td>
<td>$\sim A$</td>
</tr>
<tr>
<td>Conjunction</td>
<td>$A \land B$</td>
<td>$A &amp; B$</td>
</tr>
<tr>
<td>Disjunction</td>
<td>$A \lor B$</td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>$A \rightarrow B$</td>
<td>$A \supset B$</td>
</tr>
<tr>
<td></td>
<td>$A \Rightarrow B$</td>
<td></td>
</tr>
<tr>
<td>Biconditional</td>
<td>$A \leftrightarrow B$</td>
<td>$A \equiv B$</td>
</tr>
<tr>
<td></td>
<td>$A \iff B$</td>
<td></td>
</tr>
</tbody>
</table>

Note also that some authors refer to the conditional (or equivalently the ‘material conditional’) instead as **material implication**. This is conceptually misleading terminology, since it conflates a propositional connective in the **object language** (if-then) with the much stronger notion of **implication** or **logical consequence**, which, as we shall see, is properly expressed at the level of the **meta-language**.

§3. Propositional Syntax

3.1 Grammar and Syntax

The sentences of a language like English are, to a first approximation, **sequences of expressions**. But, because you understand English, you know, and essentially without explicitly thinking about the matter, that some sequences of expressions are **grammatical**, such as,

(1) Peter wrote a book called *Bacon’s Philosophy*

while you also know, against almost without thinking, that there are other sequences (of exactly the same expressions) which are **non-grammatical**, such as,

(2) wrote *Bacon’s* called book a *Peter Philosophy*

This capacity is called **Grammatical Competence**. It appears to be a unique feature of human beings. No other known creature exhibits this capacity.

We can give lots of examples of grammatical sequences and ungrammatical ones:

<table>
<thead>
<tr>
<th>Grammatical Sequences</th>
<th>Ungrammatical Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is easy to please John</td>
<td>Easy to is John it please</td>
</tr>
<tr>
<td>I can make you feel good</td>
<td>Feel you make good I</td>
</tr>
<tr>
<td>John is happy to teach maths</td>
<td>John are happy to teach maths</td>
</tr>
<tr>
<td>No one understands why I am miserable</td>
<td>Understands I miserable why am no one</td>
</tr>
<tr>
<td>If you smoke, you will die of cancer</td>
<td>Smoke die of will cancer you if</td>
</tr>
</tbody>
</table>

It is clear that the expression sequences on the left are grammatical and meaningful. The expression sequences on the right look like gobbledygook, and possess no straightforward meaning. Notice that you do not need to know if a sentence is true in order to know if it is meaningful.
The study of which sequences of expressions are grammatical is a major topic in modern linguistics. The main guiding idea of modern theoretical linguistics is that there is, for each language L, a set of rules that define which sequences of expressions are grammatical.

**Definition:** The grammar (or syntax) of a language L consists in a set of rules that define the set of grammatically correct expressions and formulas of L.

This approach to studying the structure of languages has been applied very successfully in this century in trying to understand the grammar (or syntax) of natural languages, like English, Spanish and so on. (See Steven Pinker 1994, *The Language Instinct*).

### 3.2 Phrase Structure Diagrams

Consider the English sentence “Cherie loves Tony”. You can analyse its grammatical structure as follows:

```
Cherie loves Tony
/   \
Noun Phrase   Verb Phrase
|   /   \
|   |   |
|   Verb   Noun Phrase
|   |   |
|   Cherie   loves   Tony
```

A diagram like this is called a **phrase structure diagram (or tree)**. Such diagrams play a central role in modern theoretical linguistics, in analysing the grammatical structure of sentences.

### 3.3 Recursive Syntax for a (Small!) Fragment of English

A phrase structure diagram shows how sentences are “built” out of more basic grammatical “building blocks”: i.e., nouns, adjectives, verbs and so on.

To make the idea more precise, consider a language L whose basic expressions are:

- **Noun Phrases:** “Tony”, “Cherie”,
- **1-Place Verb Phrases:** “walks”, “talks”,
- **2-Place Verb Phrase:** “loves”.

The grammatical rules for this language L are:

**Rule 1:** if N is a noun phrase and V is a 1-place verb, then N^V is a sentence (where N^V is the ‘concatenation’ or stringing together of N and V).

**Rule 2:** if N_1 and N_2 are noun phrases and V is a 2-place verb, then N_1^V^N_2 is a sentence.

**Rule 3.** If S is a sentence, then S is of the form N^V, where V is 1-place, or of the form N_1^V^N_2, where V is 2-place.

If you follow these rules, you can deduce that the sequences of expressions:

```
Tony talks,
```
Cherie walks,
Tony loves Cherie,
Cherie loves Tony,
are all sentences.
You can also deduce that the sequence of expressions
walks talks Tony Cherie
is not a sentence (of this language).
A system of rules, such as Rules 1, 2 and 3 above, is sometimes called a **recursive (compositional) syntax**. These rules show
(a) how the **sentences of a language are structured**,  
(b) how sentences are **built up** from primitive “**sub-atomic**” expressions (e.g., names, nouns, verbs, adjectives, etc.).

### 3.4 The Structure of Artificial Languages

Mathematical logic requires precisely specified **artificial languages**, which serve as the object systems in which arguments can be expressed and manipulated formally, and about which various meta-logical claims can be asserted and proved. This is a step of idealization (and regimentation!) necessary to obtain rigorous mathematical results. The tie to natural language and reasoning is that relevant fragments of natural languages can then be ‘translated’ into the artificial systems to obtain precise mathematical results for those particular fragments. Thus a primary goal of logic is to develop progressively richer and more powerful systems in which we can **formalize** ever larger fragments of **natural language and reasoning**.

At the moment, we are focusing on **propositional languages**. What exactly is a propositional language?

A propositional language \( L \) is specified by two sets:

(i) a set of **“atomic” sentence letters**: e.g., \{P, Q, R, \ldots\}

(ii) a set of **propositional connectives**: e.g., \{\texttt{\neg}, \land, \lor, \rightarrow, \leftrightarrow\}

To indicate the propositional language with set \( \Delta \) of sentence letters and set \( \Sigma \) of connectives, we write \( L[\Delta; \Sigma] \).

For example, if \( L \) has sentence letters \{P, Q, R\} and connectives \{\texttt{\neg}, \land\} we write:

\[
L[P, Q, R; \texttt{\neg}, \land]
\]

(when dealing with small sets we often omit set notation braces ‘\{ \}’)

For any artificial language \( L \), we give some general rules—called the **rules of syntax** or the **formation rules**—defining what it is to be a **formula of \( L \)**.

As an example we can consider the very simple propositional language \( L[P; \texttt{\neg}] \), which contains only one sentence letter and one connective.

The formation rules for \( L[P, \texttt{\neg}] \) are:

(i) if \( A \) is a sentence letter of \( L \), then \( A \) is a formula of \( L \);

(ii) if \( A \) is a formula of \( L \), then \( \texttt{\neg} A \) is a formula of \( L \);
(iii) nothing else is a formula of $\mathbf{L}$.

These rules of syntax determine for any expression containing $\neg$ and $P$ whether it is a formula of $\mathbf{L}$. According to the rules, the formulas of $\mathbf{L}[P; \neg]$ are given by the sequence.

$$P, \neg P, \neg \neg P, \neg \neg \neg P, \ldots$$

For example let us prove that the expression $\neg \neg \neg P$ is a formula of $\mathbf{L}[P; \neg]$. To prove this, first note that $P$ is formula of $\mathbf{L}$. So $\neg P$ is a formula, so $\neg \neg P$ is a formula, so $\neg \neg \neg P$ is formula of $\mathbf{L}$. QED.

[Harder] Let us also prove that the expression $\neg P\neg$ is not a formula of $\mathbf{L}[P; \neg]$. To prove this indirectly (or by reductio ad absurdum), assume the opposite. Namely, assume that the expression $\neg P\neg$ is a formula of $\mathbf{L}$. Then, from (iv) we infer that either $\neg P\neg$ is a sentence letter or $\neg P\neg = \neg B$ for some formula $B$ in $\mathbf{L}$. But $\neg P\neg$ is not a sentence letter of $\mathbf{L}$. So, $\neg P\neg$ must be $\neg B$, for some formula $B$ of $\mathbf{L}$. Thus, it follows that $B$ has the form $P\neg$, and must be a formula of $\mathbf{L}$. From (iv) we infer that either $P\neg$ is a sentence letter (but again this is false) or that $P\neg$ has the form $\neg C$, for some formula $C$ in $\mathbf{L}$. But this is impossible, because the expression $P\neg$ begins with the symbol $P$, so it cannot be the same as $\neg C$ for any expression $C$. By reductio ad absurdum then, we conclude that $\neg P\neg$ is not a formula of $\mathbf{L}$. QED.

Notice that even though $\mathbf{L}[P; \neg]$ is a very simple language, there are infinitely many formulas of this language.

3.5 Syntax for our Canonical Language of Propositional Logic

The vocabulary of a propositional language is given by its sentence letters and its propositional connectives.

**Definition:** The formal language $\mathbf{L}[\Delta; \Sigma]$ is the language whose set of sentence letters is $\Delta$ and whose set of logical connectives is $\Sigma$.

In this section we will specify a language $\mathbf{L}[P, Q, R, S \ldots; \neg, \wedge, \vee, \rightarrow, \leftrightarrow]$ that will serve as our standard or canonical language for propositional logic. Hence in future, if we refer to the propositional language $\mathbf{L}$, without explicitly mentioning the sets $\Delta$ and $\Sigma$, then in most cases (which should be clear from context) we will mean the language $\mathbf{L}[P, Q, R, S \ldots; \neg, \wedge, \vee, \rightarrow, \leftrightarrow]$.

The list $P, Q, R, S \ldots$ specifies the sentence letters of the language $\mathbf{L}[P, Q, R, S \ldots; \neg, \wedge, \vee, \rightarrow, \leftrightarrow]$. The three dots are meant to indicate that the list of sentence letters is unending. This is accomplished via the convention that the letters can occur with or without numerical subscripts. So more explicitly (and tediously), the sentence letters of the language are given by the infinite list:

$$P, Q, R, S, P_1, Q_1, R_1, S_1, P_2, Q_2, R_2, S_2 \ldots$$

It is important in our canonical language for propositional logic that we have an infinite supply of sentence letters – that way we will never run out.

The formation rules for the language $\mathbf{L}[P, Q, R, \ldots; \neg, \wedge, \vee, \rightarrow, \leftrightarrow]$ are as follows:

(i) if $A$ is a sentence letter of $\mathbf{L}$, then $A$ is a formula of $\mathbf{L}$;
(ii) if A is a formula of L, then \( \neg A \) is a formula of L;

(iii) if A and B are formulas of L, then \((A \land B), (A \lor B), (A \rightarrow B)\) and 
\((A \leftrightarrow B)\) are formulas of L;

(iv) nothing else is a formula of L.

As noted previously, parentheses or brackets are needed to indicate syntactical groupings and so distinguish between, e.g. \((P_1 \rightarrow (R_2 \land S))\) and \(((P_1 \rightarrow R_2) \land S)\), which are two distinct formulas of L generated by the foregoing rules. However, the outer set of parentheses in the foregoing examples do no real work, and so we will adopt the convention that when a formula’s main connective is binary (the intuitive notion of main connective will be covered momentarily) the outer set can be dropped to reduce clutter.

Note on the interplay between object language and metalanguage: in the formation rules for our language L of propositional logic, object language expressions are constructed with symbols in the vocabulary of L, such as ‘\(\neg\)’ and ‘\(\land\)’ and ‘\(P_2\)’, while other symbols, such as ‘A’ and ‘B’ are used as variables in the metalanguage to talk about arbitrary formulas of L. But in clauses (ii) and (iii) of the formation rules, the two types of symbols appear together, as in the expression ‘\(\neg A\)’. In such hybrid contexts, where it appears that the object language expression is being mentioned while the metavariable is being used, it is especially convenient to adopt our previously stated view that the object language symbols such as ‘\(\neg\)’ are used in the metalanguage as proper names for themselves.

3.6 Recursive Definitions of Syntax

The formation rules for L\([P, Q, R, S, \ldots; \neg, \land, \lor, \rightarrow, \leftrightarrow]\) are parallel to those of the simpler language L\([P, \neg]\) considered in the last section, and fit the general pattern of a recursive definition of the concept “formula of L”.

A recursive definition first requires a base step, in this case clause (i), which specifies an initial stock of atomic formulas as the base of the recursion (viz. the sentence letters). Clauses (ii) and (iii) are the recursion rules: they specify how to construct compound or molecular formulas by taking already specified formulas as input and yielding longer and structurally more complex formulas as output. In turn, these output formulas can then be fed back into the recursion clauses as inputs to yield ever more complex formulas, ad infinitum…

Finally, clause (iv) is the closure or exclusion clause. This is necessary for a mathematically rigorous definition of the set of formulas of L, since clauses (i)-(iii) only specify what is a formula, but they do not explicitly say what is not a formula. So without clause (iv), there would be nothing to prevent, e.g., Maggie Thatcher, the Eiffel Tower, and the expression ‘\(\land \land P_3\)’, from being included in our set of formulas, along with all the grammatically correct strings generated by (i)-(iii).

As a consequence of (i)-(iv), we can deduce that:

(v) if A is a formula of L then

(a) either A is a sentence letter of L;
(b) or A is ¬B for some formula B of L;
(c) or A is B ∧ C, for some formulas B and C of L;
(d) or A is B ∨ C, for some formulas B and C of L;
(e) or A is B → C, for some formulas B and C of L;
(f) or A is B ↔ C, for some formulas B and C of L;

So the rules (i)-(iv) determine exactly what expressions are formulas of L.

Note on the distinction between ‘infinite’ and ‘no finite upper bound’: the recursive formation rules generate an infinite number of formulas, even in the case of the relatively impoverished language L[P, ¬] possessing only one sentence letter and one connective. So our canonical language possesses an infinite number of formulas, and there is no longest formula (where length is determined by counting the number of primitive symbols it contains): for any formula A of length n, there is a formula B of length > n. So there is no finite upper bound on the length of formulas. However, no sentence of our propositional language is infinitely long! Every formula is composed of a finite number of sentence letters and connectives.

3.7 The Main Connective and Subformulas

Consider the propositional language

L[P, Q, R; ¬, ∧, ∨, →].

The formulas of this language are expressions such as,

P
¬P
¬P → (P ∧ Q)
(P ∨ R) ∨ ¬(P → Q)
and so on

It should be obvious how to extract the “main connective” for each of these formulas. Here are some simple examples:

A sentence letter, such as P, has no main connective, because it is atomic.

The main connective of the formula ¬P is ¬.

The main connective of the formula ¬P → (P ∧ Q) is the connective →.

Note that, in a fully-bracketed notation, this formula should be written

(¬P → (P ∧ Q))

It is the result of joining ¬P and P ∧ Q by a conditional.

The formulas ¬P and P ∧ Q are called subformulas of the formula ¬P → (P ∧ Q).

Similarly, the main connective of the formula (P ∨ R) ∨ ¬(P → Q) is the disjunction ∨.
The subformulas are \( P \lor R \) and \( \neg(P \rightarrow Q) \). These also have further subformulas.

**Identifying the main connective of a formula (and its subformulas) is absolutely essential in determining truth tables and in constructing our semantic tableaux later.**

### 3.8 Parsing Trees

We introduced the idea of a “phrase structure diagram” for a sentence of English. There is an analogous idea for propositional languages, called a **Parsing Tree**. The idea is that we repeatedly break down the formula into its subformulas. Obviously, we shall eventually stop.

The notion of an **immediate subformula** is recursively defined as follows:

**Definition (Immediate Subformula):**

Consider our canonical language for propositional logic \( L[P, Q, R, S \ldots; \neg, \land, \lor, \rightarrow, \leftrightarrow] \). Then:

(i) Sentence letters \( P, Q, R, S \ldots \) have no immediate subformulas;

(ii) The immediate subformula of a negation formula \( \neg A \) is the formula \( A \);

(iii) If a formula has the form \( A \land B \) or \( A \lor B \) or \( A \rightarrow B \), or \( A \leftrightarrow B \) then the immediate subformulas are \( A \) and \( B \).

To obtain the parsing tree of any given formula \( A \) you write down the formula and then underneath, you write down a tree indicating successively all the immediate subformulas.

For example, consider the formula:

\[
(P \rightarrow \neg Q) \leftrightarrow (\neg P \land (R \lor \neg \neg Q)).
\]

Its parsing tree is this:

\[
\begin{array}{c}
(P \rightarrow \neg Q) \\
\quad \leftrightarrow \\
\quad (\neg P \land (R \lor \neg \neg Q)) \\
\quad \quad / \\
\quad P \rightarrow \neg Q \\
\quad \quad \neg P \land (R \lor \neg \neg Q) \\
\quad \quad / \\
\quad P \\
\quad \quad / \\
\quad \quad \neg Q \\
\quad \quad / \\
\quad \quad Q \\
\end{array}
\]

Observe that the “endpoints” of the branches in the parsing tree for a formula are the **sentence letters** out of which the formula is composed. These sentence letters have no immediate subformulas.
It is an important fact that every parsing tree is **finite**. Each branch in the tree has only **finitely** many “nodes”. A parsing tree cannot just keep getting bigger without end. This is because, as noted earlier, every formula of \( \mathbf{L} \) has finite length, and is composed of finitely many occurrences of sentence letters and connectives.

**N.B. DO NOT CONFUSE PARSING TREES WITH SEMANTIC TABLEAUX.**

3.9 An Important Fact About Formulas

Recall the consequence (\( \Rightarrow \)) that followed from the formations rules of the propositional language \( \mathbf{L} \) above:

If \( A \) is a formula, then

(a) either \( A \) is a sentence letter;
(b) or \( A \) is \( \neg B \) for some formula \( B \);
(c) or \( A \) is \( B \land C \), for some formulas \( B \) and \( C \);
(d) or \( A \) is \( B \lor C \), for some formulas \( B \) and \( C \);
(e) or \( A \) is \( B \rightarrow C \), for some formulas \( B \) and \( C \);
(f) or \( A \) is \( B \leftrightarrow C \), for some formulas \( B \) and \( C \).

Let us first define the notion of a literal. This is simple.

A literal is either a sentence letter or the negation of one.

It is possible to show that the definition of “formula” implies the following more complicated rule:

If \( A \) is a formula, then

(a) either \( A \) is a literal; or
(b) \( A \) is \( \neg \neg B \) for some formula \( B \); or
(c1) \( A \) is \( B \land C \), for some formulas \( B \) and \( C \); or
(c2) \( A \) is \( \neg (B \land C) \), for some formulas \( B \) and \( C \); or
(d1) \( A \) is \( B \lor C \), for some formulas \( B \) and \( C \); or
(d2) \( A \) is \( \neg (B \lor C) \), for some formulas \( B \) and \( C \); or
(e1) \( A \) is \( B \rightarrow C \), for some formulas \( B \) and \( C \); or
(e2) \( A \) is \( \neg (B \rightarrow C) \), for some formulas \( B \) and \( C \); or
(f1) \( A \) is \( B \leftrightarrow C \), for some formulas \( B \) and \( C \); or
(f2) \( A \) is \( \neg (B \leftrightarrow C) \), for some formulas \( B \) and \( C \).

This tells us what the structure of every formula must be. It is either a literal (a sentence letter or the negation of one) or it is a double negation; or it is a conjunction or the negation of one; or it is a disjunction or the negation of one; and so on.

When we introduce the semantic tableau rules, we shall include a tableau rule for each of these possibilities.
§4. Propositional Semantics

4.1 Note on Sets

A set is a collection of ‘things’. The things in the collection are called its members or elements.

The “collection” containing Clinton and Diana is written:

{Clinton, Diana}.

The set containing just the objects $a$, $b$ and $c$ is written:

{a, b, c}

The order doesn’t matter. So, \{a, b, c\} is the same set as \{b, a, c\}, and so on.

We indicate that an object $a$ is a member of a set $\Delta$ by writing:

$a \in \Delta$.

We indicate the set of all cats by the following notation:

{ $x$ : $x$ is a cat },

to be read:

*the set of all things $x$ such that $x$ is a cat.*

We shall use the Greek symbols ‘$\Delta$’ and ‘$\Sigma$’ to stand for arbitrary sets.

The members of a set can be almost anything you like: physical, abstract or imaginary. So sets can contain numbers, eggs, ideas, formulas. A set can also contain other sets. (In standard set theory, a set cannot contain itself!)

One particularly important set is the empty set, which is written:

$\emptyset$, or sometimes \{ \}.

Another important set is the unit set of a single object $a$, written:

{ $a$ }.

There are two important operations on sets: $\cap$ (intersection) and $\cup$ (union):

(i) $\Delta \cap \Sigma$ the set of all things that are both in $\Delta$ and $\Sigma$
(ii) $\Delta \cup \Sigma$ the set of all things that are in either $\Delta$ or $\Sigma$.

In particular, if nothing is both in $\Delta$ and $\Sigma$, then $\Delta \cap \Sigma$ is $\emptyset$. Sometimes we shall be interested in sets of the form $\Delta \cup \{A\}$, where $\Delta$ is a set of formulas and $A$ is some formula. The set $\Delta \cup \{A\}$ is obtained by simply “adding” $A$ to the set $\Delta$. 
4.2 Validity

As stated previously, logic can be thought of as the science of valid arguments. We have already spent some time examining arguments and their structure. So now the obvious question is ‘What is validity?’ It is important to note at the outset that validity is an abstract, normative notion, not an empirical or descriptive one. As a first approximation, validity can be informally defined as follows:

**Informal Definition of Validity**

*An argument (sequent) is valid just in case it is impossible for all the premises to be true and the conclusion false.*

This definition guarantees that it’s impossible for the premises to be true and the conclusion false. However, this informal definition is somewhat vague, because we have not explained what is meant by “it is impossible”.

A clearer way to express this definition is as follows:

An argument (sequent) is valid just in case there is no possible situation in which all the premises are true and the conclusion is false.

This is clearer. Notice that it refers to “possible situations”. An equivalent way of expressing this is to say,

An argument (sequent) is valid just in case, for any possible situation, if the premises are true in that situation, then the conclusion is true in that situation.

So, we need to provide some sort of analysis of what is meant by a “possible situation” and what it means to say that a sentence is true in a situation.

In logic, the notion of a “possible situation” is analysed using the idea of an interpretation of the language in which the premises and conclusion are expressed. In the case of propositional logic, the interpretation will be an assignment of truth values to the atomic sentence letters.

So, we get a final analysis of validity:

**Definition of Validity**

*An argument (sequent) is valid just in case, for any assignment/interpretation, if the premises are true in this assignment/interpretation, then the conclusion is true as well.*

And this is equivalent to saying that,

**Definition of Validity*  

*An argument (sequent) is valid just in case there is no assignment/interpretation in which all the premises are true and the conclusion is false.*

4.3 Consistency

By far the two most important concepts introduced in any logic course are the semantical notions of validity (of an argument or of a sequent) and consistency (of a set of sentences), which, as we shall see, are intimately related.

As in the case of validity, we will give a first approximation of the notion of consistency with an informal definition:
Informal Definition of Consistency

*A set of sentences is consistent just in case it is possible for all the sentences to be true.*

And the notion of consistency can be given a more precise definition as follows:

Definition of Consistency

*A set of sentences is consistent just in case there is an assignment/interpretation in which all the sentences are true.*

4.4 Semantics of Propositional Logic

An important feature of the logical connectives is how they determine the **truth values** of formulas built from them. The basic **laws of truth for negation** \( \neg \) are:

(1a) If \( A \) is **true** (in any situation), then \( \neg A \) is **not true** (in that situation);
(1b) If \( A \) is **not true** (in any situation), then \( \neg A \) is **true** (in that situation).

These are called the “**semantic rules for negation**”. These rules (1a) and (1b) can be combined into a single rule:

(2) A negation \( \neg A \) is **true** (in any situation) if and only if \( A \) is **not true** (in that situation).

For the moment, we shall suppress the qualifier “in any situation”, and concentrate on the semantic rules for the various logical connectives.

Analogous semantic rules can be given for the propositional connectives \( \land \) and \( \lor \), as follows:

(3) A conjunction \( A \land B \) is **true** if and only if \( A \) is true and \( B \) is true.
(4) A disjunction \( A \lor B \) is **true** if and only if either \( A \) is true or \( B \) is true.

The semantic rules for \( \rightarrow \) and \( \leftrightarrow \) are a little more complicated. They are as follows,

(5) A conditional \( A \rightarrow B \) is **true** if and only if either \( A \) is **not true** or \( B \) is true.
(6) A biconditional \( A \leftrightarrow B \) is **true** if and only if \( A \) and \( B \) are both true, or neither are true.

The most peculiar of these is (5), which gives the semantic rule for the conditional \( \rightarrow \).

4.5 Non-Contradiction and Bivalence

In classical propositional logic, formulas may have only **one** of two possible “**truth values**”. Each formula must be **either true** or **false**; and no formula may be both true and false. These values are written \( T \) and \( F \). (Some authors use 1 and 0 instead. Nothing hinges on this.)

This gives rise to two further rules which always apply in classical logic:

(LNC) No formula is both true and false.  **(Law of Non-Contradiction.)**
(BIV) Every formula is either true or false.  **(Law of Bivalence.)**

In short, classical propositional logic therefore obeys non-contradiction and bivalence.
Under this assumption, we get the conclusion that

\[ A \text{ is false iff } A \text{ is not true.} \]

### 4.6 Truth-Value Assignments (for Propositional Logic)

In the above rules, we have suppressed the qualifier “in any situation”. We now explain this notion in terms of **truth-value assignments**. That is, the rules are more correctly expressed as follows:

\[(T_{\neg}) \quad \neg A \text{ is true in an assignment iff } A \text{ is false in that assignment.}\]

\[(T_{\wedge}) \quad A \wedge B \text{ is true in an assignment iff } A \text{ and } B \text{ are true in that assignment.}\]

\[(T_{\vee}) \quad A \vee B \text{ is true in an assignment iff either } A \text{ is true or } B \text{ is true in that assignment.}\]

\[(T_{\rightarrow}) \quad A \rightarrow B \text{ is true in an assignment iff either } A \text{ is false or } B \text{ is true in that assignment.}\]

\[(T_{\leftrightarrow}) \quad A \leftrightarrow B \text{ is true in an assignment iff } A \text{ and } B \text{ have the same truth value in that assignment.}\]

Next, we need to define the idea of a (truth-value) **assignment** more carefully. Intuitively, a truth value assignment corresponds to a **row** of truth values assigned to sentence letters, thus:

\[
\begin{array}{ccc}
P & Q & R \\
\text{Assignment} & F & F & T \\
\end{array}
\]

This corresponds to the “possible situation” where P is false, Q is false and R is true.

More exactly,

**Definition:** A **truth-value assignment** for a propositional language is a function which assigns to each sentence letter a truth value (i.e., either T or F).

Thus, for example, for a language with three sentence letters P, Q, and R there are exactly 8 (\(= 2^3\)) assignments.

We can then list all the assignments to, say, two sentence letters P and Q.

\[
\begin{array}{cc}
P & Q \\
\text{Assignment 1} & T & T \\
\text{Assignment 2} & T & F \\
\text{Assignment 3} & F & T \\
\text{Assignment 4} & F & F \\
\end{array}
\]

As we shall see in a moment, each distinct assignment corresponds exactly to a **row** in a truth table.

### 4.7 Truth Tables

An assignment specifies the truth values of the sentence letters.

Furthermore, given the truth values of the sentences letters, we can always **calculate** the truth values of any formula. The underlying reason is that propositional logic is **semantically compositional**.
Semantic Compositionality

A propositional language is semantically compositional because given an assignment of truth values to sentence letters, the truth value of any compound formula is also determined. That is, the truth values of compound formulas are determined by the truth values of their “components”.

This is an extremely important idea in semantic theory. There are several versions of compositionality which also apply to natural languages. If natural languages were not compositional it would be very hard to learn them.

One version of compositionality applies to meanings rather than to truth values. To illustrate, consider the sentence:

Yesterday, I saw a fnoffle.

You may almost know what this sentences means, but not know what “fnoffle” means. If you learn what this means, then you can determine what the whole sentence means. So, the meaning of a sentence depends upon the meanings of its parts.

Now, truth value compositionality is easier to understand. It means that

The truth value of a formula depends only on the truth values of its parts.

So, if you are given a formula, say \((P \land Q) \rightarrow (\neg Q \lor R)\), and I tell you the truth values of its sentence letters (i.e., the letters P, Q and R), then it is a matter of mathematical computation to find out the truth value of the whole formula.

In general, the truth value of a formula is a mathematical function of the truth values of its component sentence letters. It is this function which is expressed by the formula’s truth table. For this reason, propositional logic is often called truth-functional logic, and the propositional connectives are called truth-functional connectives.

The semantic rules above, \((T,\), \((T,\), etc., all express the semantic compositionality of propositional logic. These rules can be re-expressed very simply, using the notion of truth tables.

4.7.1 Negation

The truth table for negation is very simple:

<table>
<thead>
<tr>
<th>A</th>
<th>\¬A</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

We are concerned with a single formula \(A\), and thus two possible truth values: \(T\) and \(F\). This table indicates what we said before about the semantics of propositional logic: if a formula \(A\) is true, then its negation \(\neg A\) is false; and if \(A\) is false, then its negation \(\neg A\) is true.
4.7.2 Conjunction

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ∧ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

This indicates that a conjunction is true only when both conjuncts are true; otherwise, the conjunction is false.

4.7.3 Disjunction

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ∨ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

This indicates that a disjunction is true when either (or both) of the disjuncts are true. Because we include the case where both conjuncts are true, it is called Inclusive Disjunction. So, a disjunction is false only when both disjuncts are false.

4.7.4 Conditional

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

This is the strangest truth table, and sometimes causes confusion. It is possible to justify this strange truth table, but there is a great deal of controversy as to whether the “if…then” construction in ordinary English (or any natural language) should be analysed as above.

Roughly, a conditional is false only when its antecedent is true and its consequent is false: otherwise it is true. In particular, it is true when both components are true, and it is always true when the antecedent is false.

We shall return to the justification for this truth table below.
4.7.5 Biconditional

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ↔ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

This is very easy to justify. A biconditional “A if and only if B” should count as true just when A and B are both true, or when they are both false.

4.8 Some Examples

Equipped with this information, you can write down truth tables for any compound formula A. It is rather like calculating the values of function, such as $n^2$, and again this is why this is called truth-functional logic.

**Example 1:** consider the formula $\neg(P \land Q) \rightarrow Q$. Let’s find its truth table:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P ∧ Q</th>
<th>¬(P ∧ Q)</th>
<th>¬(P ∧ Q) → Q</th>
<th>¬(P ∧ Q) → Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Sometimes we prefer to write just the final column, thus,

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>¬(P ∧ Q) → Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

**Example 2.** Let’s do a more complicated example.

Consider the formula $(P \land Q) \rightarrow \neg(R \rightarrow (P \rightarrow Q))$. This formula contains three sentence letters and will require eight rows. Thus, we get
\[(P \land Q) \rightarrow \neg (R \rightarrow (P \rightarrow Q))\]

The values in the main column are underlined. It runs:

\[F, F, T, T, T, T, T, T\]

The most obvious thing to think about is to think about those formulas which have a main column made only of T’s and those formulas who main column consists only of F’s. The first sort of formula is called a tautology. The second is called a contradiction.

Example 3: Consider the very simple formula \(P \lor Q\). Let us write down the truth table.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>(P \land Q) \rightarrow \neg (R \rightarrow (P \rightarrow Q))</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T F F T T T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T F F T T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F T T F F F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F T F T F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F T T T T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F T T T T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F T T T T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F T T T T</td>
</tr>
</tbody>
</table>

4.8 On the Truth Table for the Conditional →

As we noted above, the truth table for → looks like this:

\[
\begin{array}{ccc}
P & Q \\
\text{Assignment} & T & T \\
\text{Assignment} & T & F \\
\text{Assignment} & F & T \\
\end{array}
\]

Observe that there are three assignments in which the formula \(P \lor Q\) is true. These assignments can be indicated thus:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How is this justified?

Clearly, if \( A \) is true, and \( B \) is false, then “if \( A \) then \( B \)” should be false. So, the second line in the truth table for \( \rightarrow \) seems perfectly justified.

However, why should “if \( A \), then \( B \)” be true when \( A \) is false?

And why should “if \( A \) then \( B \)” be true when both \( A \) and \( B \) are true?

**There is, in fact, a large philosophical literature concerning the semantics of conditionals, and it is a matter of dispute whether “if … then” is properly expressed by its standard classical truth table above.**

For this reason, the truth table is usually called the truth table of the *material conditional*, \( \rightarrow \).

However, the above truth table is a very natural consequence of what has to be the case in order to validate certain kinds of mathematical reasoning involving conditionals.

Consider the following statement:

*Any square of an even number is an even number.*

As we shall see later, this statement can be analysed more fully, as,

*For any object \( x \), if \( x \) is the square of an even, then \( x \) is even.*

This is called a **quantified conditional**. (The initial phrase ‘for any object \( x \)’, is called a *logical quantifier*. We examine these later in the course, when we turn to Predicate Logic.)

And let us rewrite this as follows,

*For any object \( x \), \((x \text{ is the square of an even}) \rightarrow (x \text{ is even})\).*

Now, surely this statement is **true**.

And since it is true, **each of its instances** must be true.

(That is, if a universal statement is true, then every instance must be true.)

So, we can consider various values of the variable ‘\( x \)’: the numbers 0, 1, 2, 3, etc.

\((0 \text{ is the square of an even}) \rightarrow (0 \text{ is even}).\)

Note that both the antecedent and consequent are true. And the whole thing is, as agreed, true.

But consider compositionality. The truth value of the whole thing can only depend upon the truth values of the parts. So, it follows that, in general:
If the antecedent and consequent of a conditional are both true, then the whole conditional must also be true.

This gives us the first line in the above truth table.

Similarly,

$$(1 \text{ is the square of an even}) \rightarrow (1 \text{ is even})$$

In this case, the antecedent is false, and the consequent is false. So, in general:

If the antecedent and consequent of a conditional are both false, then the conditional is true.

This corresponds to the final line in the truth table.

Finally,

$$(2 \text{ is the square of an even}) \rightarrow (2 \text{ is even})$$

In this case, the antecedent is false, and the consequent is true. So, in general:

If the antecedent of a conditional is false and the consequent true, then the conditional is true.

This corresponds to the third line in the truth table.

The above argumentation provides just one way of justifying why the “if … then” connective should have the truth table given above. If one is still worried about this, simply be aware that, in this logic course, $\rightarrow$ always means the material conditional, and the truth table for the material conditional is the one given above.

§5. Basic Logical Concepts

5.1 Summary So Far

You are now equipped with the notions of:

(i) **sentence letters:** $P, Q, R, S, \ldots$

(ii) **propositional connectives:** $\neg, \land, \lor, \rightarrow, \leftrightarrow$

(iii) **formulas:** $P \land P, \neg Q, Q \rightarrow (Q \rightarrow P)$, and so on

(iv) **truth value assignments.**

(iv) **truth tables.**

With these ideas, you can define some central logical concepts.

5.2 Tautology

Some statements are logically trivial. They are automatically true, irrespective of the situation you consider. Examples are:

Either the earth is flat or the earth is not flat.

If you are happy, then you are happy.

You are Spanish if and only if you are Spanish.

If 7 is prime and 7 is an odd number, then 7 is an odd number.

In logic, these correspond to what are called tautologies.
Definition 1. A formula $A$ is a **tautology** just in case $A$ is true for any assignment of truth values to sentence letters in $A$.

Examples:

$P \lor \neg P$.  \hspace{1cm} (aka, the Law of Excluded Middle)

$P \leftrightarrow P$

$(P \land Q \rightarrow P)$

5.3 Contradiction

Some statements are logically trivial in the sense that they cannot be true. They are automatically or necessarily false, irrespective of the situation you consider. Examples are:

The earth is flat if and only if the earth is not flat.

England is larger than France and England is not larger than France.

In logic, these correspond to what are called **contradictions**.

Definition 2. A formula $A$ is a **contradiction** just in case $A$ is false, for any assignment of truth values to sentence letters in $A$.

Examples:

$P \land \neg P$

$P \leftrightarrow \neg P$

$(P \lor Q) \land (\neg Q \land \neg P)$

5.4 Logical Contingency

Most statements may be either true or false. They are not necessarily true, and not necessarily false. We call such statements **contingent**. In the case of logic, we call formulas which may be true, and may be false, **logically contingent**.

Definition 3. A formula $A$ is **logically contingent** just in case $A$ is true in at least one assignment and false in at least one assignment.

Examples:

$P \land Q$

$P \leftrightarrow (\neg P \land Q)$

$(P \land Q) \rightarrow (Q \lor R)$

5.5 Logical Equivalence

Another important phenomenon is that we can have some statement $A$ which is **equivalent** to some other statement $B$. Roughly, in whatever situation you consider, $A$ is true if and only if $B$ is true. So, $A$ and $B$ have the same truth value, in any situation. Examples are

‘It is not the case that snow is white and grass is orange’ is equivalent to ‘Either snow is not white or grass is not orange’.

‘If that is justice, then I am a banana’ is equivalent to ‘Either that is not justice, or I am a banana’.

In logic, this relationship is called **logical equivalence** and is defined as follows:

Definition 4. A formula $A$ is a **logically equivalent** to a formula $B$ just in case $A$ and $B$ have the same truth values in any assignment.
We shall use the symbol \( \equiv \) to indicate logical equivalence.

**Examples:**

\[
P \rightarrow Q \quad \equiv \quad \neg P \lor Q
\]
\[
\neg (P \land Q) \quad \equiv \quad \neg P \lor \neg Q
\]

Some equivalences are very obvious. For example

\[
P \land Q \quad \equiv \quad Q \land P
\]
order doesn’t matter

\[
P \lor Q \quad \equiv \quad Q \lor P
\]
order doesn’t matter

\[
P \land (Q \land R) \quad \equiv \quad (P \land Q) \land R
\]
brackets don’t matter for \( \land \).

\[
P \lor (Q \lor R) \quad \equiv \quad (P \lor Q) \lor R
\]
brackets don’t matter for \( \lor \).

Compare with + and \( \times \) in arithmetic: 3 + 6 is equal to 6 + 3. And 3 + (6 + 12) is equal to (3 + 6) + 12. Similarly, 3 \( \times \) 6 is equal to 6 \( \times \) 3, and 3 \( \times \) (6 \( \times \) 12) is equal to (3 \( \times \) 6) \( \times \) 12. We say that the addition function + is **commutative** and **associative**.

These equivalences are very useful. Instead of writing

\[
((P \land Q) \land (Q \rightarrow R)) \land S
\]

We can write more simply:

\[
P \land Q \land (Q \rightarrow R) \land S
\]

**5.6 Consistency**

Just as important as the notion of validity is the notion of **consistency**. Intuitively, a sentence is consistent if there is a possible situation in which it is true. And a set of sentences is consistent if there is a possible situation in which all the sentences are true.

As we shall see below, the notions of validity and consistency are closed related.

**Definition 5.** A formula \( A \) is **consistent** just in case there is an assignment in which \( A \) is true.

**Definition 6.** A set \( \Delta \) of formulas is **consistent** just in case there is an assignment such that every formula in \( \Delta \) is true in that assignment.

**5.7 A Model of a Set of Formulas**

Suppose that \( \Delta \) is a set of formulas, and that this set is consistent. So, there is an assignment such that, for every formula \( A \) in \( \Delta \), \( A \) is **true** in this assignment. Any such assignment is called a **model** of \( \Delta \).
**Definition 7.** A **model** of a set $\Delta$ of formulas is an assignment such that every formula in $\Delta$ is **true** in this assignment.

Clearly, a set $\Delta$ of formulas is **consistent** just in case it has a model.

**Examples:**

Let $\Delta$ be the set $\{P, Q\}$

A model of $\Delta$ is the assignment:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

Let $\Delta$ be the set $\{\neg P, Q, P \rightarrow Q\}$

A model of this set is the assignment:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

### 5.8 Inconsistency

**Definition 8.** A set $\Delta$ of formulas is **inconsistent** just in case there is **no** assignment in which every formula in $\Delta$ is true.

**Examples:**

$\Delta = \{P, \neg P\}$

$\Delta = \{\neg P, \neg Q, P \lor Q\}$

$\Delta = \{P \rightarrow Q, P, \neg Q\}$

Each of the above sets is inconsistent. None has a model.

### 5.9 Logical Consequence

**Definition 9.** A formula $A$ is a **logical consequence** of a set of formulas $\Delta$ just in case any assignment that makes all the formulas in $\Delta$ true, also makes $A$ true.

**Notation:** We introduce a special symbol ‘$\vdash$’ and we shall write:

$$\Delta \vdash A$$

to be read:

**A is a logical consequence of the set of formulas $\Delta$.**

or

**$\Delta$ semantically implies $A$.**

**Examples:**

$\{P \land Q\} \vdash P$.

$\{P, Q\} \vdash P$.

$\{P \rightarrow Q, P\} \vdash Q$. 
Another note on levels: the symbol ‘⊨’ belongs to the **metalanguage**. It is **not** a symbol of our propositional object language **L**, and it is **not** to be confused with the object level connective ‘→’ (though as we shall see momentarily, there **is** an important **relation** between the two).

5.10 Validity of Arguments

Consider an argument, with the typical structure,

[Premise 1]

[Premise 2]

...

Therefore, [Conclusion]

Then,

**Definition 10.** An argument is **valid** just in case there is no assignment that makes all the premises true and the conclusion false.

In other words, an argument is valid if and only if the conclusion is a logical consequence of the premises. This is equivalent to the definition that we gave earlier.

5.11 Validity of Sequents

Consider a sequent, of the form:

\[ A_1, A_2, ..., A_n : B \]

or of the form

\[ \Delta : A \]

Then,

**Definition 11.** A sequent is **valid** just in case there is no assignment that makes all the premises true and the conclusion false.

This corresponds to the definition that we gave earlier.

5.12 Relationships Among These Logical Concepts

There are several relationships between the logical concepts. Consider a sequent

\[ \Delta : A \]

This is valid just in case there is no assignment such that every formula in \( \Delta \) is true and \( A \) is false. This is equivalent to saying that \( \Delta \) semantically implies \( A \) or that \( A \) logically follows from \( \Delta \).

So, the sequent is valid just in case there is no assignment in which every formula in \( \Delta \) is true, and \( \neg A \) is also true. So, this means that the sequent is valid just in case there is no assignment in which every formula in \( \Delta \cup \{ \neg A \} \) is true. Recalling the definition of “consistent”, it follows that:

**The sequent** \( \Delta : A \) **is valid just in case the set** \( \Delta \cup \{ \neg A \} \) **is inconsistent.**
This is a conceptually fundamental relationship that will be explored in more detail in the next section.

Similarly, we can easily show that

The sequent $\Delta : A$ is valid just in case $\Delta \models A$.

We can show several other relationships between these concepts:

- $A$ is a contradiction just in case $\neg A$ is a tautology.
- $A$ is a tautology just in case $\neg A$ is a contradiction.
- $A \equiv B$ just in case the formula $A \leftrightarrow B$ is a tautology.
- $A_1, \ldots, A_n \models B$ just in case the formula $(A_1 \land \ldots \land A_n) \rightarrow B$ is a tautology.

§6. Introducing Semantic Tableaux

6.1 Introducing Tableaux

From now on we shall be studying sequents of the form

$A_1, A_2, \ldots, A_n : B$

And we shall sometimes write

$\Delta : A$

(where $\Delta$ is the set of premises of the sequent).

We want to devise a method for classifying which sequents are valid and which are invalid.

The fundamental result we shall use is based on the point noted at the end of the previous section:

The sequent $A_1, \ldots, A_n : B$ is valid just in case the set $\{A_1, \ldots, A_n, \neg B\}$ is inconsistent.

IT EXTREMELY IMPORTANT TO UNDERSTAND THIS IDEA.

It expresses a fundamental relationship between the notions of validity and inconsistency, and is in most cases the most convenient way of thinking about validity in a technical context.

**Proof:** From our definition of validity, the sequent $A_1, \ldots, A_n : B$ is valid just in case there is no assignment in which $A_1, \ldots, A_n$ are all true and $B$ is false. So, the sequent is valid just in case there is no assignment in which all of $A_1, \ldots, A_n, \neg B$ are true. So, the sequent is valid just in case the set $\{A_1, \ldots, A_n, \neg B\}$ is inconsistent (has no model).

**Definition:** Given a sequent $A_1, \ldots, A_n : B$, we call the set $\{A_1, \ldots, A_n, \neg B\}$ the counter-example set.
More generally, for a sequent $\Delta : A$, the counter-example set is $\Delta \cup \{\neg A\}$.

A technique for demonstrating validity of a sequent will thus be equivalent to a technique for demonstrating inconsistency of the counter-example set. This is tantamount to proving that the sequent (or corresponding argument) has no counter-example, which in turn is another way of saying that it is valid.

So,

**We are looking for a method which shows that a given set of formulas is inconsistent.**

We shall devise a method, called the **Semantic Tableau Method**, which, given an initial set of formulas, may (or may not) result in the conclusion that the set is inconsistent. The sign of this will be that the **tableau is closed**.

First, note that it is obvious that the set $\{A, \neg A\}$ is inconsistent. Indeed, if a set contains any formula $A$ and its negation $\neg A$, then it must be inconsistent. Call any such pair a **contradictory pair**.

Second, consider a list of formulas written **vertically**, for example:

```
P
Q \land R
R
\neg Q \rightarrow P
\neg P
```

Notice that this list contains both $P$ and $\neg P$. So, it is definitely inconsistent.

Our method will similarly involve lists of formulas, and if they contain a contradictory pair of formulas $A$ and $\neg A$, then obviously the set is inconsistent.

**First Informal Example:** Consider the following set of formulas: $\{P \land Q, \neg P\}$. It is easy to show using a truth table that this set is inconsistent. However, we can show that it is inconsistent **without using a truth table**, by a form of deductive reasoning. That is, by following computer-programmable inference rules. This reasoning generates various lists of formulas which all contain a contradictory pair. As you will see, these lists look like branches of an upside down tree.

Consider the following reasoning:

```
Suppose:  1.  P \land Q is true     initial assumption
and  2.  \neg P is true     initial assumption

| then,  3.  P is true  [from 1]
and  4.  Q is true  [from 1]
```

Since we are always saying, “Suppose $P$ is true”, we can just eliminate the predicate “is true” (for supposing that $P$ is true is equivalent to simply supposing $P$), and re-write the reasoning thus:
Suppose:
1. \( P \land Q \) initial assumption
and
2. \( \neg P \) initial assumption

then,
3. \( P \) [from 1]
and
4. \( Q \) [from 1]

We have inferred a contradiction, because line (2) contains \( \neg P \), while line (3) contains \( P \), and obviously this is a contradictory pair.

From this we conclude that the initial set \( \{ P \land Q, \neg P \} \) is inconsistent.

**Second Informal Example**: Consider the set \( \{ P \lor Q, \neg P, \neg Q \} \). Reason as follows:

Suppose:
1. \( P \lor Q \) initial assumption
and
2. \( \neg P \) initial assumption
and
3. \( \neg Q \) initial assumption

so,
4. \( P \lor Q \) [from 1]

In this, the possibilities “branch”, because the assumption \( P \lor Q \) is a disjunction. So, we get two “branches”. But note that each branch contains a contradictory pair. The left-hand branch contains \( \neg P \) and \( P \), while the right-hand branch contains \( \neg Q \) and \( Q \).

We may therefore conclude that the initial set \( \{ P \lor Q, \neg P, \neg Q \} \) is inconsistent.

For later comparison, let us now write out again these examples of reasoning:

<table>
<thead>
<tr>
<th>Informal Example 1</th>
<th>Informal Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( P \land Q ) initial assumption</td>
<td>1. ( P \lor Q ) initial assumption</td>
</tr>
<tr>
<td>2. ( \neg P ) initial assumption</td>
<td>2. ( \neg P ) initial assumption</td>
</tr>
<tr>
<td>3. ( P ) [from 1]</td>
<td>3. ( \neg Q ) initial assumption</td>
</tr>
<tr>
<td>4. ( Q ) [from 1]</td>
<td>4. ( P \lor Q ) [from 1]</td>
</tr>
</tbody>
</table>

Notice that when we reason with a conjunction, our branch is extended downwards. When we reason with a disjunction, we generate two further branches.

6.2 The Underlying Idea of Semantic Tableaux

The basic idea of a semantic tableau is to consider some initial list \( \Delta \) and investigate all possibilities for its being true. So, we assume that \( \Delta \) is true. By deductive reasoning, this leads to various possibilities. We obtain various conclusions: the formulas taken together constitute a branch. In a sense, a branch represents the description of a possible state of affaires, or possible world. In some cases (e.g., when we consider a disjunction formula of the form \( A \lor B \)), we generate two branches. These represent two different possible worlds. And further branches may be generated.
Now, what may happen as we apply our reasoning is that a branch will generate a formula $A$ and its negation $\neg A$. Obviously, these cannot both be true (it is impossible for $A$ and $\neg A$ to both be true). So, this is not a genuinely possible world. We say that the branch is closed. Now if every branch from the initial list is closed, there is no possible world in which the initial list $\Delta$ is true. So, we conclude that the initial list $\Delta$ must be inconsistent.

Schematically, we have:

$$\text{Closed tableau } \Rightarrow \text{ initial list is inconsistent}$$

(This is called the SOUNDERNESS property of semantic tableaux.)

What also may happen is that a branch may eventually finish without containing any contradictions. That is, for every formula in the branch, we have applied all of the tableau rules and we have not obtained any contradiction. Such a branch is called open and finished. In this case, we can use the list of formulas in the branch to construct a branch model, which is an assignment which satisfies the initial list $\Delta$. Thus, we may conclude that the initial list $\Delta$ is consistent.

Schematically, we have:

$$\text{Finished and open tableau } \Rightarrow \text{ initial list is consistent}$$

(This is called the COMPLETENESS property of semantic tableaux.)

In this introductory course, we do not expect you to study the proof of these two results: Soundness and completeness. But we shall use them frequently.

6.3 Formalizing Tableaux

Consider the first informal example of a tableau above. We formalize it like this:

| 1. $P \land Q$ | initial list |
| 2. $\neg P$ | initial list |
| | |
| 3. $P$ | [from 1]. |
| 4. $Q$ | [from 1]. |

The above structure is a semantic tableau (also called a semantic tree, or a truth tree). Notice that there is just one branch, and it contains $\neg P$ and $P$, a contradictory pair. Thus it is closed. (This is what the symbol $\blacksquare$ indicates.)

This shows that the initial list $\{P \land Q, \neg P\}$ is inconsistent.

The second example looked like this,

| 1. $P \lor Q$ | initial list |
| 2. $\neg P$ | initial list |
| | |
| 3. $\neg Q$ | initial list |
| | |
| 4. $P$ $Q$ | [from 1] |
This is another semantic tableau. There are two branches, and both contain contradictory pairs. This shows that the initial list \{P ∨ Q, \neg P, \neg Q\} is inconsistent.

6.4 Tableau Rule for Conjunctions

In the first tableau pattern, we had the general sub-pattern:

\[
\begin{align*}
A \land B \\
| \\
.... \\
A \\
B
\end{align*}
\]

(The specific case was \(P \land Q\). But we can reason the same way even if the conjuncts are not sentence letters. So, we can reason this way whatever the formulas \(A\) and \(B\) are.)

I.e., from any conjunction \(A \land B\), we inferred both its conjuncts, \(A\) and \(B\).

This rule just summarizes the information in the truth table that,

\textit{if }A \land B\textit{ is true, then }A\textit{ is true and }B\textit{ is true.}

But we have ignored the following information from the truth table, that

\textit{if }A \land B\textit{ is false, then either }A\textit{ is false or }B\textit{ is false.}
We can include this information by writing a **pair of branches**:

\[
\neg (A \land B) \\
\text{/} \\
\neg A \quad \neg B
\]

Notice that the **negation** of a conjunction generates two extra branches. As you may recall, the formula \(\neg (A \land B)\) is logically equivalent to the conjunction \(\neg A \lor \neg B\), and this is why we get the two branches. We have formed two branches, indicating **two possibilities**.

Now we can put these two tableau patterns together as follows:

<table>
<thead>
<tr>
<th>Tableau Rule for (\land)</th>
<th>Tableau Rule for (\neg \land)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A \land B)</td>
<td>(\neg (A \land B))</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>(A)</td>
<td>(\neg A) \quad \neg B</td>
</tr>
<tr>
<td>(B)</td>
<td></td>
</tr>
</tbody>
</table>

These are the **tableau rules for conjunction and negated conjunction**.

The first **rule** shows what follows if \(A \land B\) is **true**.

The second **rule** shows what follows if \(A \land B\) is **false**.

### 6.5 The Tableau Rules

We will see that, except for the rule for double negation, the tableau rules come in pairs. Each pair of tableau rules can be deduced from the corresponding truth table. The only exception is the rule for \(\neg \neg\), which has just one rule rather than a pair. But this is very simple.

The reason why the tableau rules come in pairs is connected to our earlier result that every formula is either a literal, or a double negation, or a conjunction or the negation of one, or a disjunction or the negation of one, and so on.

#### 6.5.1 Double Negation:

The tableau rule for **double negation** (a formula beginning \(\neg \neg\)) is simplicity:

\[
\neg \neg A \\
\text{|} \\
A
\]

This formalizes the inference,

\(\neg \neg A\) is **true**. Thus, \(A\) is **true**.
6.5.2 Conjunction:

We have seen these before:

\[
\begin{array}{c|c}
A \land B & \neg(A \land B) \\
\hline
A & \neg A \\
B & \neg B \\
\end{array}
\]

These formalize the inferences:

- \(A \land B\) is true. Thus, A is true and B is true.
- \(\neg(A \land B)\) is true. Thus, either \(\neg A\) is true or \(\neg B\) is true.

6.5.3 Disjunction:

Similar tableau diagrams can be constructed for disjunction formulas. Thus

\[
\begin{array}{c|c}
A \lor B & \neg(A \lor B) \\
\hline
A & \neg A \\
B & \neg B \\
\end{array}
\]

These formalize the inferences:

- \(A \lor B\) is true. Thus, either A is true or B is true.
- \(\neg(A \lor B)\) is true. Thus, \(\neg A\) is true and \(\neg B\) is true.

6.5.4 Conditional (\(\rightarrow\))

We can work out the tableau rule for \(\rightarrow\) from the truth table, which informs us that

- if \(A \rightarrow B\) is true, then either A is false or B is true.
- if \(\neg(A \rightarrow B)\) is true, then A is true and B is false.

We can then put the corresponding diagrams together as follows:

**Tableau Rules for \(\rightarrow\)**

\[
\begin{array}{c|c}
A \rightarrow B & \neg(A \rightarrow B) \\
\hline
\neg A & B \\
A & \neg B \\
\end{array}
\]

The easiest way to remember this is to remember that

The formula \(A \rightarrow B\) is logically equivalent to the formula \(\neg A \lor B\).

(i.e., \(A \rightarrow B \equiv \neg A \lor B\))

And thus, the tableau for \(\rightarrow\) should be the same as given by this. And we get
\[ \neg A \lor B \quad \neg \neg (\neg A \lor B) \]
\[ / \quad \backslash \quad / \quad \backslash \]
\[-A \quad B \quad \neg \neg A \]
\[-B \quad \neg B \quad A \]

And these are the same as we had before.

### 6.5.5 Biconditional (\(\leftrightarrow\))

A similar analysis reveals the following rules for \(\leftrightarrow\):

<table>
<thead>
<tr>
<th>Tableau Rules for (\leftrightarrow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A \leftrightarrow B) \quad \neg (A \leftrightarrow B)</td>
</tr>
<tr>
<td>/ \quad \backslash \quad / \quad \backslash</td>
</tr>
<tr>
<td>(A\quad \neg A\quad A\quad B)</td>
</tr>
<tr>
<td>(B\quad \neg B\quad \neg B\quad \neg A)</td>
</tr>
</tbody>
</table>

The easiest way to remember this is that

*The formula \(A \leftrightarrow B\) is logically equivalent to \((A \land B) \lor (\neg A \land \neg B)\).*

I.e., \(A \leftrightarrow B = (A \land B) \lor (\neg A \land \neg B)\).

If you write the tableau for this formula, it will be equivalent to the above.

### 6.6 Semantic Tableau Rules for Propositional Logic: Summary

<table>
<thead>
<tr>
<th>Double Negation ((\neg \neg))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\neg \neg A)</td>
</tr>
<tr>
<td>/ \quad \backslash</td>
</tr>
<tr>
<td>(A)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conjunction ((\land)) \quad Negated Conjunction ((\neg \land))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A \land B) \quad \neg (A \land B)</td>
</tr>
<tr>
<td>/ \quad \backslash</td>
</tr>
<tr>
<td>(A\quad \neg A\quad \neg B)</td>
</tr>
<tr>
<td>(B)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disjunction ((\lor)) \quad Negated Disjunction ((\neg \lor))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A \lor B) \quad \neg (A \lor B)</td>
</tr>
<tr>
<td>/ \quad \backslash</td>
</tr>
<tr>
<td>(A\quad \neg A\quad \neg B)</td>
</tr>
<tr>
<td>(B)</td>
</tr>
<tr>
<td>Conditional ($\rightarrow$)</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>/\</td>
</tr>
<tr>
<td>$\neg A$ $\quad B$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Biconditional ($\leftrightarrow$)</th>
<th>Negated Biconditional ($\neg \leftrightarrow$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \leftrightarrow B$</td>
<td>$\neg (A \leftrightarrow B)$</td>
</tr>
<tr>
<td>/\</td>
<td>/\</td>
</tr>
<tr>
<td>$A \quad \neg A$</td>
<td>$A \quad \neg A$</td>
</tr>
<tr>
<td>$B \quad \neg B$</td>
<td>$\neg B \quad B$</td>
</tr>
</tbody>
</table>

Using these nine simple algorithmic rules you can construct semantic tableaux which allow you to:

(i) **prove** any **valid** sequent;

(ii) construct a **counter-example** for any **invalid** sequent.

§7. Tableau Methods I

7.1 The Method of Semantic Tableaux

The idea of a semantic tableau is to investigate the consistency/inconsistency of an initial list $\Delta$ of formulas.

The idea is, that by applying the tableau rules, we eventually produce a branch containing a **contradictory pair** $\neg A$ and $A$, and we call the branch **closed** (note that $A$ need not be a sentence letter). When that happens, it indicates an **inconsistency** in that branch. If all of the branches in a tableau lead to inconsistency, then we have shown that the initial list is **inconsistent**.

On the other hand, if a branch does not contain a contradictory pair, we say that the branch is **open**. Furthermore, if all the formulas on a branch have been “used”, then the branch is **finished**. So if a finished branch contains no contradictory pairs and is finished, we call it **finished and open**. In this case, we can construct an assignment
which satisfies the initial list, and thus we can conclude that the initial list is **consistent**.

Let us summarize the terminology.

1. **Closed (Branch)**
   
   If a branch contains a contradictory pair (a formula $A$ and its negation $\neg A$), then the branch is **closed**.

2. **Open (Branch)**
   
   If a branch does not contain any contradictory pair, we say it is **open**.

3. **Used (Formula)**
   
   When an inference rule is applied to a formula, we say that the formula has been **checked** or **used**.

4. **Finished (Branch)**
   
   When every formula in a branch has been used, we say that the branch is **finished**.

5. **Closed (Tableau)**
   
   If **every** branch of a tableau is closed, we say that the **tableau is closed**.

6. **Open (Tableau)**
   
   If a tableau contains a finished and open branch, we say that the **tableau is open**.

7. **Completed (Tableau)**
   
   When every branch is either closed or finished, the **tableau is completed**.

Finally we add some further terminology:

8. **Children**
   
   If a rule is applied to a formula $A$, generating new formulas $B$, $C$, $\ldots$, then these are called the **children** of the formula $A$.

   For example, the children of any conjunction $A \land B$ are $A$ and $B$. The children of $A \rightarrow B$ are $\neg A$ and $B$ (so the children of a formula **might not** coincide with its **immediate subformulas**). And so on. Of course, the children may themselves generate further children (i.e., grand-children), and further great-grand-children and so on. We may call these **descendants**. However, it is a basic fact about the formulas of propositional logic that given any finite initial list $\Delta$, there are only finitely many descendants possible, before you reach sentence letters. So, semantic tableaux in propositional logic are always finite. This, however, is not true in predicate logic.
7.1.1 Flow Diagram for Constructing Semantic Tableaux

Write down the formulas of the initial list.

Pick a formula in an open branch and apply one of the tableau rules and write down the result, adding **lines numbers** and **annotations**.

Put a tick (♀) next to the formula used.

Examine any branch. If it contains a formula A and its negation ¬A, then place ♀ after the final formula in the branch. That branch is **closed**.

Is every branch closed?

**YES**

STOP
The tableau is closed.

**NO**

Has every formula in every branch been used?

**YES**

STOP: The tableau is finished and open.

**NO**
The foregoing is a method for constructing completed tableaux. This is necessary when proving that an initial list is inconsistent, and hence the corresponding sequent is valid, because it must be shown that there is no possible counterexample. However, when proving that an initial list is consistent, and the corresponding argument is invalid, the search need not be exhaustive, since just one finished, open branch is proof that the initial list has a model.

7.2 Showing that a Set is Inconsistent

Example 1:
Suppose our initial list is Δ = {P → Q, P, ¬Q}. We want to show that Δ is inconsistent. This is how you proceed:

<table>
<thead>
<tr>
<th>line number</th>
<th>tableau</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>P → Q</td>
<td>✓</td>
</tr>
<tr>
<td>2.</td>
<td>P</td>
<td>Δ: initial list</td>
</tr>
<tr>
<td>3.</td>
<td>¬Q</td>
<td></td>
</tr>
<tr>
<td>/ \</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>¬P Q</td>
<td>→ 1</td>
</tr>
</tbody>
</table>

Examine this tableau carefully.

We first list the initial set Δ as lines (1) to (3).

Then we picked line (1) and applied the \(\rightarrow\) rule to obtain line (4), which now contains two branches. We ticked line (1).

Then we examined the left branch and saw that it was closed (the branch contains P and ¬P). So, we underline the last formula. That branch is closed.

Then we examined the right branch and saw that it was closed too (it contains ¬Q and Q). So that branch was closed.

Now, every branch is closed. So, the whole tableau is closed.

(Observe that we place ‘■’ under the final formula in any closed branch).

Example 2:
Is the set Δ = {P ∨ Q, ¬P, ¬Q} consistent?

Here’s the semantic tableau for Δ:

<table>
<thead>
<tr>
<th>line number</th>
<th>tableau</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>P ∨ Q</td>
<td>✓</td>
</tr>
<tr>
<td>2.</td>
<td>¬P</td>
<td>Δ: initial list</td>
</tr>
<tr>
<td>3.</td>
<td>¬Q</td>
<td></td>
</tr>
<tr>
<td>/ \</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>P Q</td>
<td>∨ 1</td>
</tr>
</tbody>
</table>

Every branch is closed. Hence, the tableau for Δ is closed. Hence, Δ is inconsistent.
Example 3:

Is the set $\Delta = \{(P \lor Q) \rightarrow R, \neg R \land (P \lor Q)\}$ consistent?

Here’s the tableau:

1. $(P \lor Q) \rightarrow R$ \checkmark \quad initial list $\Delta$
2. $\neg R \land (P \lor Q)$ \checkmark \quad initial list $\Delta$
   
3. $\neg R$ \quad $\land$ 2
4. $P \lor Q$ \checkmark \quad $\land$ 2
   
5. $\neg (P \lor Q)$ \checkmark $R$ \quad $\rightarrow$ 1
   
6. $\neg P$ \quad $\neg \lor$ 5
7. $\neg Q$ \quad $\neg \lor$ 5
   
8. $P$ \quad $Q$ \quad $\lor$ 4
   
---

Every branch is closed. Hence, the tableau is \textbf{closed}. Hence, the initial list $\Delta$ is \textbf{inconsistent}.

Note: When you apply a tableau rule to a formula to extend a branch or generate new branches, these extensions or new branches must be added to \textbf{every branch} containing that formula.

7.3 Tableau Proof

We have now arrived at a point where we can explain a central concept. This is the concept of a \textbf{tableau proof}. Let us define this precisely.

\textbf{Definition 1:} Suppose that $\Delta$ is a finite set of formulas and that $A$ is a formula. Then we say that \textbf{there is a tableau proof of $A$ from $\Delta$} just in case there is a closed tableau whose initial list is $\Delta \cup \{\neg A\}$.

We shall write:

$\Delta \vdash A$

to mean:

there is a \textbf{tableau proof} of $A$ from the set $\Delta$.

The symbol ‘$\vdash$’ is sometimes called the symbol for the \textbf{deducibility relation}. The reason is that sometimes instead of saying that there is a tableau proof of $A$ from $\Delta$, we say instead that

$A$ is \textbf{provable} from $\Delta$, or that,
A is **deducible** from $\Delta$.

When we have a finite list of formulas, we usually omit the set brackets { and }, and write e.g.,

$$P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R$$

instead of

$$\{P \rightarrow Q, Q \rightarrow R\} \vdash P \rightarrow R$$

### 7.4 Examples

**Example 4.** We show that $P, P \rightarrow Q \vdash Q$

1. $P$ initial list
2. $P \rightarrow Q$ initial list
3. $\neg Q$ initial list

\[\begin{array}{l}
4. \neg P \quad Q \rightarrow 2 \\
\end{array}\]

(Notice that we have dropped the ticks, $\checkmark$. We’ll reintroduce them later.)

The tableau is closed: there is a tableau proof of $Q$ from $\{P, P \rightarrow Q\}$.

So, $\{P, P \rightarrow Q\} \vdash Q$.

**Example 5:** We show that $P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R$

1. $P \rightarrow Q$ initial list
2. $Q \rightarrow R$ initial list
3. $\neg(P \rightarrow R)$ initial list

\[\begin{array}{l}
4. P \quad \neg \rightarrow 3 \\
5. \neg R \quad \neg \rightarrow 3 \\
\end{array}\]

\[\begin{array}{l}
6. \neg P \quad Q \rightarrow 1 \\
\end{array}\]

\[\begin{array}{l}
7. \neg Q \quad R \rightarrow 2 \\
\end{array}\]

### 7.5 An Important Point About Applying the Tableau Rules

Let us illustrate some points about semantic tableau which are slightly confusing to begin with. Consider the previous example again.

$$P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R.$$ But we apply the rules in a different order, thus:
1. \( P \rightarrow Q \) initial list
2. \( Q \rightarrow R \) initial list
3. \( \neg(P \rightarrow R) \) initial list

/ \\  
4. \( \neg P \) \( Q \) \rightarrow 1  
    / \\  
5. \( P \) \( P \) \rightarrow 3  
6. \( \neg R \) \( \neg R \) \rightarrow 3  

/ \  
7. \( \neg Q \) \( R \) \rightarrow 2  
    / \  

Notice that when we apply the \( \neg \rightarrow \) rule at lines 5 and 6, we must add the two formulas to both branches. I.e., we must add \( P \) and \( \neg R \) to both branches, which is what we see at lines 5 and 6.

In other words, if a segment of branch actually branches further, to two sub-branches, and the initial segment contains a formula \( A \land B \), then we must \( A \) and \( B \) to both of these sub-branches.

The same is true if the formula we apply the rule to is itself a branching formula. For example, let us do this example again in the worst order! Thus,

1. \( P \rightarrow Q \) initial list
2. \( Q \rightarrow R \) initial list
3. \( \neg(P \rightarrow R) \) initial list

/ \  
4. \( \neg P \) \( Q \) \rightarrow 1  
    / \  
5. \( \neg Q \) \( R \) \( \neg Q \) \( R \) \rightarrow 2  
    / \  
6. \( P \) \( P \) \( P \) \rightarrow 3  
7. \( \neg R \) \( \neg R \) \( \neg R \) \rightarrow 3  

In this case, we applied \( \rightarrow \) to line 1, giving line 4. Then we applied \( \rightarrow \) to line 2, which gives line 5, but we must add two branches (\( \neg Q \) and \( R \)) twice.

So, whenever we develop the tableau, we must add the descendants of a formula to all branches which contain that formula.
7.6 Reduce Branching!

Here is a rule-of-thumb for constructing nice tableaux. Although your tableaux will turn out OK if you ignore this rule-of-thumb, they will tend to be messier.

Reduce Branching

If possible, try to apply non-branching rules before you apply branching rules.

For example, if you re-examine the three tableaux above, then the first is the nicest, while the second and third have more branching.

7.7 More Examples

We provide tableau proofs of the following:

(i) \( P \rightarrow Q \vdash \neg Q \rightarrow \neg P \)

1. \( P \rightarrow Q \) initial list
2. \( \neg (\neg Q \rightarrow \neg P) \) initial list
3. \( \neg Q \) \( \rightarrow \ 2 \)
4. \( \neg \neg P \) \( \rightarrow \ 2 \)

/ \  
5. \( \neg P \quad \neg Q \rightarrow 1 \)

/ \  

(ii) \( \neg (P \rightarrow Q) \vdash P \land \neg Q \)

1. \( \neg (P \rightarrow Q) \) initial list
2. \( \neg (P \land \neg Q) \) initial list
3. \( P \) \( \rightarrow 1 \)
4. \( \neg Q \) \( \rightarrow 1 \)

/ \  
5. \( \neg P \quad \neg \neg Q \rightarrow \land 2 \)

/ \  

(iii) \( Q \vdash P \rightarrow Q \)

1. \( Q \) initial list
2. \( \neg (P \rightarrow Q) \) initial list
3. \( P \) \( \rightarrow 2 \)
4. \( \neg Q \) \( \rightarrow 2 \)

/
(iv) \[ P \vdash Q \rightarrow (P \land Q) \]

1. \[ P \] initial list
2. \[ \neg(Q \rightarrow (P \land Q)) \] initial list
3. \[ Q \] \[\rightarrow\] 2
4. \[ \neg(P \land Q) \] \[\rightarrow\] 2

/ \[
5. \neg P \quad \neg Q \quad \neg \land 4

This is an interesting valid sequent. It classifies arguments such as the following as valid:

Texas is a large state

*Therefore*, either snow is white or snow is not white.

(v) \[ P \vdash Q \lor \neg Q \] “Anything implies a tautology”

1. \[ P \] initial list
2. \[ \neg(Q \lor \neg Q) \] initial list
3. \[ \neg Q \] \[\neg \lor\] 2
4. \[ \neg \neg Q \] \[\neg \lor\] 2

This is an interesting valid sequent. It classifies arguments such as the following as valid:

Snow is white and snow is not white.

*Therefore*, the earth is flat.

(vi) \[ P \land \neg P \vdash Q \] “A contradiction implies anything”

1. \[ P \land \neg P \] initial list
2. \[ \neg Q \] initial list
3. \[ P \] \[\land\] 1
4. \[ \neg P \] \[\land\] 1

So, it classifies arguments such as the following as valid:

Snow is white and snow is not white.

*Therefore*, the earth is flat.
7.8 Some More Complicated Examples

(i). \( P \rightarrow Q, \neg P \rightarrow Q \vdash Q \)

1. \( P \rightarrow Q \) initial list
2. \( \neg P \rightarrow Q \) initial list
3. \( \neg Q \) initial list

4. \( \neg P \quad Q \rightarrow 1 \)

5. \( \neg P \quad Q \rightarrow 2 \)

Note: we do not have to apply \( \neg \neg \) rule at line 5. For the branch contains \( \neg P \) and \( \neg \neg P \) and thus is closed.

(ii). \( P \land (Q \land R) \vdash (P \land Q) \land R \)

1. \( P \land (Q \land R) \) initial list
2. \( \neg ((P \land Q) \land R) \) initial list
3. \( P \land 1 \)
4. \( Q \land R \land 1 \)
5. \( Q \land 4 \)
6. \( R \land 4 \)

7. \( \neg (P \land Q) \neg R \neg \land 2 \)

8. \( \neg P \neg Q \neg \land 7 \)
(iii). \[ P \rightarrow R, Q \rightarrow \neg R \vdash \neg (P \land Q) \]

1. \[ P \rightarrow R \quad \text{initial list} \]
2. \[ Q \rightarrow \neg R \quad \text{initial list} \]
3. \[ \neg (P \land Q) \quad \text{initial list} \]
4. \[ P \land Q \quad \neg \neg 3 \]
5. \[ P \quad \land 4 \]
6. \[ Q \quad \land 4 \]

\[ \begin{array}{c}
\vdash \\
\end{array} \]
7. \[ \neg P \quad R \quad \rightarrow 1 \]
8. \[ \neg Q \quad \neg R \quad \rightarrow 2 \]

(iv). \[ (P \land Q) \rightarrow \neg R \vdash R \rightarrow (P \rightarrow \neg Q) \]

1. \[ (P \land Q) \rightarrow \neg R \quad \text{initial list} \]
2. \[ \neg (R \rightarrow (P \rightarrow \neg Q)) \quad \text{initial list} \]
3. \[ R \quad \neg \neg 2 \]
4. \[ \neg (P \rightarrow \neg Q) \quad \neg \neg 2 \]
5. \[ P \quad \neg \neg 4 \]
6. \[ \neg \neg Q \quad \neg \neg 4 \]

\[ \begin{array}{c}
\vdash \\
\end{array} \]
7. \[ \neg (P \land Q) \quad \neg R \quad \lor 1 \]
8. \[ \neg P \quad \neg Q \quad \neg \land 7 \]

7.9 Theoremhood

We have not considered the case where \( \Delta \) is the empty set \( \emptyset \). This is a possible (and important) situation. We use a special notation

\[ \vdash A \]

which means

\[ \emptyset \vdash A \]

I.e., there is a tableau proof of \( A \) from the empty set of premises.

I.e., the initial list is just \( \{ \neg A \} \) and the resulting tableau is closed.

**Definition 2:** When we have \( \vdash A \), we say that \( A \) is a theorem of propositional logic.
An introductory note on metatheory: as will be discussed in a little more detail in section 12, it is possible to prove that the class of theorems of propositional logic is identical to the class of tautologies. This is an important result in logical metatheory: a higher level proof about our object level system of tableau proofs. The core idea is that the syntactical system of tableau proofs is adequate to exactly mirror the relevant semantical phenomena. This metatheoretical fact will be implicitly assumed in much of section 8.

7.10 Examples

(i) \( \vdash P \rightarrow (Q \rightarrow P) \)

1. \( \neg [P \rightarrow (Q \rightarrow P)] \) initial list
2. \( P \) \( \neg \rightarrow 1 \)
3. \( \neg (Q \rightarrow P) \) \( \neg \rightarrow 1 \)
4. \( Q \) \( \neg \rightarrow 3 \)
5. \( \neg P \) \( \neg \rightarrow 3 \)

\[ \]

The tableau is closed. Thus, \( \vdash P \rightarrow (Q \rightarrow P) \).

(ii) \( \vdash ((P \rightarrow P) \rightarrow Q) \rightarrow Q \)

1. \( \neg (((P \rightarrow P) \rightarrow Q) \rightarrow Q) \) initial list
2. \( (P \rightarrow P) \rightarrow Q \) \( \neg \rightarrow 1 \)
3. \( \neg Q \) \( \neg \rightarrow 1 \)
4. \( \neg (P \rightarrow P) \) \( Q \) \( \rightarrow 2 \)
5. \( P \) \( \neg \rightarrow 4 \)
6. \( \neg P \) \( \neg \rightarrow 4 \)

\[ \]

Here are some other examples, which you might try for yourself.

(iii) \( \vdash (P \rightarrow Q) \rightarrow ((Q \rightarrow R) \rightarrow (P \rightarrow R)) \)

(iv) \( \vdash P \lor (P \rightarrow Q) \)

(v) \( \vdash (P \rightarrow Q) \lor (Q \rightarrow R) \)

(vi) \( \vdash ((\neg P \rightarrow R) \land (\neg Q \rightarrow R)) \rightarrow (\neg (P \land Q) \rightarrow R) \)
§8. Tableau Methods II

8.1 What You Can Do With a Semantic Tableau

There are several things that you can do with a semantic tableau:

(a) Demonstrate that a set is inconsistent;
(b1) Demonstrate that a sequent is valid;
(b2) Demonstrate that an argument is valid;
(b3) Demonstrate that a formula is a logical consequence of some premises;
(c1) Demonstrate that a formula is a contradiction;
(c2) Demonstrate that a formula is a tautology;
(d) Demonstrate that a pair of formulas are logically equivalent.

Now (b1), (b2) and (b3) are all essentially doing the same thing. And (a) is closely related to these in that to show that a sequent is valid is to show that the counter-example set is inconsistent. Also, (c1) and (c2) are related (please do not forget which way round to do this: it is a common mistake not to include the necessary negation when demonstrating that a formula A is a tautology.) Finally, (d) just involves showing that each formula is deducible from the other.

8.2 Demonstrating Inconsistency

We have already seen how to show that a set \( \Delta \) is inconsistent. Construct a semantic tableau for \( \Delta \) by listing the elements of \( \Delta \) and apply the tableau rules. If the tableau is closed, then \( \Delta \) is inconsistent.

**Example 1**: Consider the set \( \Delta = \{ (P \lor Q) \lor R, \neg P, \neg Q, \neg R \} \)

Here’s the tableau for \( \Delta \):

1. \((P \lor Q) \lor R\) initial list
2. \neg P\ initial list
3. \neg Q\ initial list
4. \neg R\ initial list
5. \P\ \Q\ \R\ \lor\ 1
6. \P\ \Q\ \lor\ 5

The tableau is closed. Thus \( \Delta \) is inconsistent.
8.3 Demonstrating Validity of a Sequent

Suppose you have a sequent

$$A_1, \ldots, A_n : B$$

And you want to prove that it is valid. As we noted above, to show that this sequent is valid, we need to show that the counter-example set \(\{A_1, \ldots, A_n, \neg B\}\) is inconsistent.

**DO NOT FORGET TO NEGATE THE CONCLUSION FORMULA!!!**

Example 2: We show that the sequent \(P \lor Q, \neg P : Q\) is valid.

The counter-example set is \(\{P \lor Q, \neg P, \neg Q\}\). We must show that this is inconsistent. We have done this before, but here it is again.

1. \(P \lor Q\)  \(\checkmark\)  initial list
2. \(\neg P\)  \(\}\)  (counter-example set)
3. \(\neg Q\)  \(\}\)
4. \(P \quad Q \quad \lor 1\)

Hence, the **tableau** is **closed**. Hence, the counter-example set is **inconsistent**. Hence, the sequent \(P \lor Q, \neg P : Q\) is valid.

8.4 Demonstrating Validity of an Argument

Suppose you have an argument:

Premise 1, 

\[\ldots\]

Premise \(n\)\[\]

Therefore: Conclusion

The method is effectively the same as the previous sub-section. For the argument is valid just in case the sequent

Premise 1, \(\ldots\), Premise \(n\) : Conclusion

is valid.

So, we construct a tableau for the counter-example set

\(\{\text{Premise } 1, \ldots, \text{Premise } n, \neg \text{Conclusion}\}\)

If the tableau closes, then this counter-example set is inconsistent, and thus the argument is valid.

**Example 3:** consider the argument:

If John is singing, then Yoko is happy \([\text{Premise } 1]\)

John is singing \([\text{Premise } 2]\)

Therefore, Yoko is happy \([\text{Conclusion}]\)
This is formalized thus:

\[
P \to Q  \\
P  \\
\hline
Q
\]

Therefore,

This form of reasoning is so important that it has a name: **Modus Ponens**.

The counter-example set is thus \{P \to Q, P, \neg Q\}. The appropriate tableau is then:

1. \(P \to Q\) \hspace{1cm} \text{(initial list)}
2. \(P\) \hspace{1cm} \text{(counter-example set)}
3. \(\neg Q\)
4. \(\neg P\) \hspace{1cm} \(Q\) \to 1

\[\]

**Example 4:** Consider the argument:

Either he’s a fool or I’m a banana \hspace{1cm} [Premise 1]
I am not a banana \hspace{1cm} \hspace{1cm} [Premise 2]

Therefore, \hspace{1cm} He’s a fool \hspace{1cm} [Conclusion]

This is formalized thus:

\[
P \lor Q  \\
\neg Q  \\
\hline
P
\]

The counter-example set is \{P \lor Q, \neg Q, \neg P\}. We have done this before. The tableau for this counter-example set is closed. Thus, the associated argument is valid.

Again, this is an example of an important kind of reasoning: it is called **Disjunctive Syllogism**.

**8.5 Demonstrating Logical Consequence**

Suppose you what to check if \(\Delta \vdash A\). To repeat,

\[\Delta \vdash A\] just in case \(\Delta \cup \{\neg A\}\) is inconsistent.

So, construct a tableau with initial list \(\Delta \cup \{\neg A\}\). If this is closed, then \(\Delta \vdash A\).
Example 5: To check if $P \lor Q, P \rightarrow R, Q \rightarrow R \vdash R$.

The counter-example set is $\{P \lor Q, P \rightarrow R, Q \rightarrow R, \neg R\}$. Thus,

1. $P \lor Q$  
2. $P \rightarrow R$ (initial list)  
3. $Q \rightarrow R$ (counter-example set)  
4. $\neg R$  
5. $P \lor Q$  

The tableau is closed. Thus, $P \lor Q, P \rightarrow R, Q \rightarrow R \vdash R$.

8.6 Demonstrating Contradiction

A formula $A$ is a contradiction just in case it is inconsistent. So, to show that $A$ is a contradiction, you construct a tableau from the initial list $\{A\}$. If the tableau is closed, then $A$ is a contradiction.

Example 6: Consider the formula $P \land \neg P$. Here’s the tableau:

1. $P \land \neg P$ (initial list)  
2. $P$  
3. $\neg P$  

The tableau for the formula $P \land \neg P$ is closed. Thus, $P \land \neg P$ is a contradiction.

Here’s a fancier example.
Example 7: Consider the formula \( \neg(P \rightarrow Q) \land \neg(Q \rightarrow R) \).

1. \( \neg(P \rightarrow Q) \land \neg(Q \rightarrow R) \) initial list

   | 2. \( \neg(P \rightarrow Q) \land 1 \)
   | 3. \( \neg(Q \rightarrow R) \land 1 \)

   | 4. \( P \rightarrow 2 \)
   5. \( \neg Q \rightarrow 2 \)

   | 6. \( Q \rightarrow 3 \)
   7. \( \neg R \rightarrow 3 \)

So, the formula \( \neg(P \rightarrow Q) \land \neg(Q \rightarrow R) \) is a contradiction.

8.7 Demonstrating Tautology

A formula \( A \) is a tautology just in case its negation \( \neg A \) is a contradiction. So, to show that \( A \) is a tautology, you construct a tableau from the initial list \( \{\neg A\} \). If the tableau is closed, then \( \neg A \) is a contradiction and thus \( A \) is a tautology.

Example 8: To show that the formula \( P \leftrightarrow P \) is a tautology. Take the negation, i.e., \( \neg(P \leftrightarrow P) \) and form a tableau:

1. \( \neg(P \leftrightarrow P) \) initial list

   | 2. \( P \rightarrow \neg P \leftrightarrow 1 \)
   3. \( \neg P \rightarrow P \leftrightarrow 1 \)

The tableau is closed. Thus, \( \neg(P \leftrightarrow P) \) is a contradiction. So, \( P \leftrightarrow P \) is a tautology.
Example 9: To show that the formula \((P \lor Q) \iff (\neg P \rightarrow Q)\) is a tautology.

Take the negation, i.e., \(\neg[(P \lor Q) \iff (\neg P \rightarrow Q)]\) and form a tableau:

1. \(\neg[(P \lor Q) \iff (\neg P \rightarrow Q)]\) initial list
2. \(P \lor Q\) \(\neg(P \lor Q)\) \(\neg\iff 1\)
3. \(\neg(\neg P \rightarrow Q)\) \(\neg P \rightarrow Q\) \(\neg\iff 1\)
   \| \| \|
4. \(\neg P\) \(\neg\rightarrow 3\) \(\neg P\) \(\neg \lor 2\)
5. \(\neg Q\) \(\neg\rightarrow 3\) \(\neg Q\) \(\neg \lor 2\)
   \| \| \| \|
6. \(P\) \(Q\) \(\lor 2\) \(\neg P\) \(Q\) \(\rightarrow 3\)

The tableau for the formula \(\neg[(P \lor Q) \iff (\neg P \rightarrow Q)]\) is closed.

Thus, \(\neg[(P \lor Q) \iff (\neg P \rightarrow Q)]\) is a contradiction. Thus, the formula \((P \lor Q) \iff (\neg P \rightarrow Q)\) is a tautology.

8.8 Demonstrating Logical Equivalence

We showed earlier that a formula \(A\) is logically equivalent to another \(B\) just in case the biconditional formula \(A \iff B\) is a tautology. So, to show that \(A \equiv B\), you need to show that the formula \(A \iff B\) is tautology. That is, construct a tableau whose initial formula is \(\neg(A \iff B)\). If the tableau closes, then \(A \equiv B\).

Another method uses the fact that \(A \equiv B\) just in case \(A \models B\) and \(B \models A\). To put this method into practice, you must construct two tableaux, the first to check if \(A \models B\) and the second to check if \(B \models A\). These tableaux begin thus:

\[
\begin{array}{ccc}
A & & B \\
\neg B & & \neg A \\
\end{array}
\]

If both tableaux close, then \(A \equiv B\).
**Example 10**: To show that $P \rightarrow Q \equiv \neg P \lor Q$.

We use the second method. That is, we show that $P \rightarrow Q \models \neg P \lor Q$ and $\neg P \lor Q \models P \rightarrow Q$.

Here is the first tableau.

1. $P \rightarrow Q$ initial list
2. $\neg (\neg P \lor Q)$ initial list
   
3. $\neg P$ \hspace{1cm} \neg \lor 2
4. $\neg Q$ \hspace{1cm} \neg \lor 2
   
5. $P$ \hspace{1cm} \neg \neg 3
   \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm}
6. $\neg P$ \hspace{1cm} $Q$ \hspace{1cm} \rightarrow 1
   \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm}
   \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm}

The tableau is closed. Thus, $P \rightarrow Q \models \neg P \lor Q$.

Second,

1. $\neg P \lor Q$ initial list
2. $\neg (P \rightarrow Q)$ initial list
   
3. $P$ \hspace{1cm} \neg \rightarrow 2
4. $\neg Q$ \hspace{1cm} \neg \rightarrow 2
   
   \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm}
5. $\neg P$ \hspace{1cm} $Q$ \hspace{1cm} \lor 1
   \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm}
   \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm}

The tableau is closed, and thus $\neg P \lor Q \models P \rightarrow Q$.

Hence, we may conclude that $P \rightarrow Q \equiv \neg P \lor Q$. 
§9. Tableau Methods III

9.1 What If a Tableau Contains An Open Finished Branch?

You may have noticed that we have not yet explored an important possibility.

**What if a finished tableau contains an open finished branch?**

Consider the following initial list: $\Delta = \{P \lor Q, \neg P\}$. First, construct a truth table:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P $\lor$ Q</th>
<th>$\neg$P</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

So, this set $\Delta$ has exactly one model:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

(*)

Now construct a tableau for $\Delta$ as follows:

1. $P \lor Q$ \checkmark initial list
2. $\neg P$ \checkmark initial list

/ \n
3. $P$ \checkmark $Q$ \checkmark $1$

??

The tableau is finished (each complex formula in each branch has been used). One branch in the tableau is closed. But there is also an open finished branch. This branch contains the formulas $\neg P$ and $Q$. What does this mean? The answer is simple:

**An open finished branch means that the initial list $\Delta$ is consistent:**

That is, an open finished branch means that there is a model of $\Delta$.

9.2 Branch Models & Consistency

The tableau for $\Delta = \{P \lor Q, \neg P\}$ has an open finished branch containing the formulas $\neg P$ and $Q$. If $Y$ is a finished open branch, then we can define an assignment that makes all of the formulas in the branch true. Recall, a literal is either a sentence letter or the negation of a sentence letter. In particular, consider an assignment that makes all the literals true. Thus, suppose that $\neg P$ and $Q$ are both true in some assignment. This yields the assignment as follows:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

(**)

Compare (*) with (**): this looks like a coincidence! The model we obtained from the truth table is the same as the assignment we get from the tableau.
In fact, this is no coincidence. If you run through an open branch \( Y \) in a finished tableau for an initial set \( \Delta \) of formulas and assign \( \top \) to each literal, then you determine an assignment for the sentence letters \( P, Q, \) etc. It can be generally proved that whenever you do this procedure, the resulting assignment will be a model of \( \Delta \): this assignment will make every formula in \( \Delta \) true.

In short, if a finished tableau for \( \Delta \) is open, then any open finished branch determines an assignment in which \( \Delta \) is true (i.e., a model for \( \Delta \)). Since there is such a model for \( \Delta \), it then follows that \( \Delta \) is consistent. So, if a finished tableau for \( \Delta \) is open, then \( \Delta \) is consistent.

### 9.3 Procedure for Constructing Branch Models

What is a **branch model** for a set \( \Delta \) of formulas? How do you find it? Suppose that \( L \) is a **propositional language** with sentence letters \( P, Q, R, \ldots \). Then the **literals** of \( L \) are the formulas:

\[
P, \neg P, Q, \neg Q, R, \neg R, \ldots
\]

That is, a literal is either a **sentence letter** or the **negation of a sentence letter**.

Suppose you are given a set \( \Delta \) of formulas. This is how to find a branch model for \( \Delta \):

(i) Take the initial list \( \Delta \) and construct the tableau for \( \Delta \);

(ii) When the tableau is **finished**, detect any **open branches**;

(iii) For any open branch \( Y \), write down all the **literals** in \( Y \) as a list:

(iv) Define the assignment such that each of these literals is **true**.

**Illustration**: Consider a finished open tableau for \( \Delta \) with formulas built from the letters \( P, Q \) and \( R \). Suppose the tableau has two open branches \( Y_1 \) and \( Y_2 \) with the following literals:

\[
\begin{align*}
Y_1: & \quad P, \quad \neg Q, \quad \neg R \\
Y_2: & \quad P \quad Q, \quad \neg R
\end{align*}
\]

Then the appropriate branch models are given thus:

\[
\begin{array}{ccc}
P & Q & R \\
\text{Assignment for } Y_1: & T & F & F \\
\text{Assignment for } Y_2: & T & T & F
\end{array}
\]

It is usually easy to check that these assignments will be models of the initial set \( \Delta \).

In fact, it is possible to show (at least in the case of propositional logic) that this method yields **all** the models of a consistent set.

That is, there are no other models of the initial list.
9.4 Examples

Example 1: Find all the branch models of the set \( \Delta = \{ P \rightarrow Q, Q, \neg P \} \).

1. \( P \rightarrow Q \checkmark \) initial list
2. \( Q \) initial list
3. \( \neg P \) initial list

\[
\begin{array}{c}
/ & \\
4. & \neg P & Q \rightarrow 1 \\
Y_1 & Y_2
\end{array}
\]

The tableau is finished. There are two open branches:

- \( Y_1: \neg P \quad Q \)
- \( Y_2: \neg P \quad Q \)

Notice that these branches both determine the same branch model:

\[
\begin{array}{cc}
P & Q \\
\text{Branch model} & F & T
\end{array}
\]

If you check, you will see that this assignment satisfies \( \Delta \). If you check, you will also see that there are no other assignments which satisfy \( \Delta \). So, this is the unique model of \( \Delta \).

Example 2: Find all the branch models of the set \( \Delta = \{ \neg P \leftrightarrow Q, \neg Q \} \).

1. \( \neg P \leftrightarrow Q \checkmark \) initial list
2. \( \neg Q \) initial list

\[
\begin{array}{c}
/ & \\
3. & \neg P & \neg \neg P \leftrightarrow 1 \\
4. & Q & \neg Q \leftrightarrow 1 \\
\blacksquare & | \\
5. & P & \neg \neg 3 \\
Y
\end{array}
\]

The tableau is finished and open. We have the open finished branch \( Y \):

- \( Y: P \quad \neg Q \)

This branch determines the branch model:

\[
\begin{array}{cc}
P & Q \\
\text{Branch model} & T & F
\end{array}
\]

If you check, you will see that this assignment satisfies \( \Delta \). I.e., \( \neg P \leftrightarrow Q \) and \( \neg Q \) are both true in this assignment. Again, there are no other models.
Example 3: Find all the branch models of the set \( \Delta = \{P \leftrightarrow \neg Q, Q \lor R\} \).

1. \( P \leftrightarrow \neg Q \checkmark \) initial list
2. \( Q \lor R \checkmark \) initial list

3. \( P \neg P \leftrightarrow 1 \)

4. \( \neg Q \neg \neg Q \checkmark \leftrightarrow 1 \)

\[
\begin{array}{ccc}
P & Q & R \\
\hline
\text{Y}_1 & \checkmark & | \\
\text{Y}_2 & \checkmark & | \\
\text{Y}_3 & \checkmark & | \\
\end{array}
\]

5. \( Q \neg Q \neg \neg 4 \)

This tableau is finished, with three open branches, containing the following literals:

\( \text{Y}_1: \) \( P \neg Q R \)
\( \text{Y}_2: \) \( \neg P Q \)
\( \text{Y}_3: \) \( \neg P Q R \)

The first and third branches determine the following branch models:

\[
\begin{array}{ccc}
P & Q & R \\
\hline
\text{Assignment for Y}_1 & \checkmark & \checkmark & \checkmark \\
\text{Assignment for Y}_3 & \checkmark & \checkmark & \checkmark \\
\end{array}
\]

The second branch \( \text{Y}_2 \) contains only the literals \( \neg P \) and \( Q \). In this case, any truth value assignment for \( R \) will make \( \Delta \) true. So, \( \text{Y}_2 \) determines two assignments: one with \( R \) true and one with \( R \) false. Note that branch \( \text{Y}_3 \) contains \( \neg P, Q \) and \( R \), so \( R \) must be true on this branch. So, \( \text{Y}_3 \) already determines one of these assignments. But \( R \) could also be false on \( \text{Y}_2 \), as long as \( \neg P \) and \( Q \) are true. So, the final branch model is:

\[
\begin{array}{ccc}
P & Q & R \\
\hline
\text{Assignment for Y}_2 & \checkmark & \checkmark & \checkmark \\
\end{array}
\]

Now you can check that these three assignments, are all models of the set \( \Delta = \{P \leftrightarrow \neg Q, Q \lor R\} \). That is, if you write out the truth table, you will get:

| \( P \) \( Q \) \( R \) \( P \leftrightarrow \neg Q \) \( Q \lor R \) \( \Delta \) |
|---|---|---|---|---|---|
| Assignment for \( \text{Y}_1 \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) |
| Assignment for \( \text{Y}_2 \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) |
| Assignment for \( \text{Y}_3 \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) | \( \checkmark \) |
§10. Tableau Methods IV

10.1 Proving That a Set of Formulas is Consistent

We have already explained how to prove that a set \( \Delta \) of formulas is consistent. You construct the tableau for \( \Delta \). If the finished tableau has open branches, then each open branch determines branch models which satisfy \( \Delta \). Since all formulas in \( \Delta \) are true in these assignments, it follows that \( \Delta \) is consistent.

Thus, the tableau method provides a mechanical method for:

(i) Proving that a set of formulas \( \Delta \) is inconsistent; or

(ii) Finding models of \( \Delta \) (thus showing that \( \Delta \) is consistent).

Both of these procedures involve constructing a tableau for \( \Delta \):

(i)* If the tableau for \( \Delta \) is closed, then \( \Delta \) is inconsistent.

(ii)* If a finished tableau for \( \Delta \) is open, then \( \Delta \) is consistent, and any open branch determines at least one branch model for \( \Delta \).

NOTE WELL: You cannot conclude from an unfinished open branch that it will remain open: any of its branches may still close when you use any unused formulas. Only when a branch is finished are you allowed to use this open branch to construct a branch model.

10.2 Examples

Example 1. Show that the set \{P, Q, \neg R\} is consistent.

We construct a finished open tableau. The branch model can immediately be read off.

1. P
2. Q
3. \neg R

This is a finished open tableau. The literals are P, Q and \neg R. Hence the branch model is:

\[
\begin{array}{ccc}
P & Q & R \\
T & T & F \\
\end{array}
\]

Example 2. Show that the set \{P \lor Q, Q \land R\} is consistent.

1. P \lor Q \checkmark \quad \text{initial list}
2. Q \land R \checkmark \quad \text{initial list}
3. Q \land 2
4. R \land 2

\[
/ \\
\sqrt{V} \\
\]
5. P \lor 1

This is a finished open tableau, with two finished open branches.
These branches contain the literals

\[
\begin{array}{ccc}
P & Q & R \\
Q & R
\end{array}
\]

In the first case, the branch model is obviously:

\[
\begin{array}{ccc}
P & Q & R \\
T & T & T
\end{array}
\]

In the second case, the truth value of \( P \) is not specified. So, it could be either \( T \) or \( F \). The first case is covered by the above model. So, the second case is given by

\[
\begin{array}{ccc}
P & Q & R \\
F & T & T
\end{array}
\]

**Example 3**: Show that the set \( \{P \rightarrow Q, Q, \neg P\} \) is consistent.

1. \( P \rightarrow Q \) initial list
2. \( Q \) initial list
3. \( \neg P \) initial list

\[
\begin{array}{ccc}
P & Q & R \\
F & T & T
\end{array}
\]

The tableau is finished and open. Both branches contain the same literals, \( \neg P \) and \( Q \). Hence, the branch model is

\[
\begin{array}{ccc}
P & Q \\
F & T
\end{array}
\]

Again, there are no other models.

10.3 **Proving that an Argument (or Sequent) is Invalid**

A second fundamental use of branch models is to show that an argument (or sequent) is *invalid*. As you now know, an argument with the form:

\[
\begin{align*}
& \text{Premise 1} \\
& \text{Premise 2} \\
& \ldots \\
\text{Premise } n \\
\text{Therefore: } \text{Conclusion}
\end{align*}
\]

is *valid* just in case the counter-example set \( \Delta = \{\text{Premise 1, Premise 2, …, Premise } n, \neg \text{Conclusion}\} \) is **inconsistent**.

It follows that an argument is *invalid* iff the counter-example set \( \Delta \) is **consistent**.

The set \( \Delta \) is consistent just in case there is an assignment which satisfies \( \Delta \) (this assignment can be any branch model for \( \Delta \)). Any such assignment makes all the premises true and \( \neg \text{Conclusion} \) true. Thus, any such assignment makes all the premises true and the conclusion false. An assignment that makes all the premises true and the conclusion false is called a **counter-example**.
The tableau method provides a direct way of finding counter-examples for invalid arguments.

**Example 4.** Consider the following informal argument:

If John is singing, then Yoko is happy

Yoko is happy

Therefore, John is singing

Obviously, this argument is **invalid**.

To demonstrate this, you must first **formalize**:

\[
P \rightarrow Q \\
Q \\
\therefore P
\]

The associated counter-example set \( \Delta \) is \{\( P \rightarrow Q \), \( Q \), \( \neg P \)\}. We have already constructed the tableau for this set and found the branch model that makes \( \Delta \) true:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

If you check you will see that with this assignment, all the premises (i.e., the formulas \( P \rightarrow Q \) and \( Q \)) are true and the conclusion (i.e., the formula \( P \)) is false. So, this assignment is a counter-example to the argument.

A counter-example is thus an assignment for the formulas in a formal argument such that all the premises are true and the conclusion is false.

**Example 5:** Consider the argument

If you go, I will go

Therefore, Unless you go, I will not go

This formalized thus

\[
P \rightarrow Q \\
\therefore P \lor \neg Q
\]

Is this a valid argument? The counter-example set \( \Delta \) is \{\( P \rightarrow Q \), \( \neg (P \lor \neg Q) \)\}. Then,

1. \( P \rightarrow Q \checkmark \) initial list
2. \( \neg (P \lor \neg Q) \checkmark \) initial list
   |  
3. \( \neg P \) \( \neg \lor 2 \)
4. \( \neg \neg Q \checkmark \) \( \neg \lor 2 \)
5. \( Q \) \( \neg \neg 4 \)
   / \  
6. \( \neg P \) \( Q \) \( \rightarrow 1 \)

The tableau is finished. Both branches are still open.
Both branches contain the literals \{\neg P, Q\}. Hence, the counter-example is the following assignment:

\[
\begin{array}{c|c|c}
P & Q & \text{Branch model} \\
\hline
F & T & \end{array}
\]

You may check that the premise \(P \to Q\) is true, but the conclusion \(P \lor \neg Q\) is false in this assignment. Because there is a counter-example, the argument must be invalid.

### 10.4 More Examples

Let us use the tableau method to show that a couple of other sequents are invalid:

**Example 6: **Show that the sequent

\[
P \to Q, Q \to R : R \to P
\]

is invalid.

The counter-example set is \{\(P \to Q, Q \to R, \neg(R \to P)\}\}. Here is the tableau:

1. \(P \to Q\) \(\checkmark\) initial list
2. \(Q \to R\) \(\checkmark\) initial list
3. \(\neg(R \to P)\) \(\checkmark\) initial list
4. \(R\) \(\neg \to 3\)
5. \(\neg P\) \(\neg \to 3\)
6. \(\neg P\) \(Q\) \(\to 1\)
7. \(\neg Q\) \(R\) \(\neg Q\) \(R\) \(\to 2\)

There are three finished open branches, determining two branch models:

\[
\begin{array}{c|c|c|c}
P & Q & R & \text{Branch model} \\
\hline
F & F & T & \\
F & T & T & \\
\end{array}
\]

So, the sequent is invalid. If \(P\) and \(Q\) are false, and \(R\) is true, then the premises are true and the conclusion is false. Also, if \(P\) is false, and \(Q\) and \(R\) are true, then the premises are true and the conclusion is false.
**Example 7:** Show that the sequent  

$$(P \land Q) \rightarrow R : P \rightarrow R$$

is invalid.

The counter-example set is $\{(P \land Q) \rightarrow R, \neg(P \rightarrow R)\}$. Here is the tableau:

1. $(P \land Q) \rightarrow R \checkmark$ initial list
2. $\neg(P \rightarrow R) \checkmark$ initial list
3. $P \neg \rightarrow 2$
4. $\neg R \neg \rightarrow 2$

$$\begin{array}{c}
\text{5. } \neg(P \land Q) \checkmark \quad R \quad \rightarrow 1 \\
\text{6. } \neg P \quad \neg Q \quad \neg \land 5
\end{array}$$

There is one finished open branch, with literals $P, \neg Q, \neg R$, thus determining the branch model:

$$\begin{array}{ccc}
P & Q & R \\
\text{Branch model} & T & F & F
\end{array}$$

So, the sequent is invalid. If $P$ is true, and $Q$ and $R$ are false, then the premises are true and the conclusion is false. As noted, this is the only counter-example to this invalid sequent.

Let us just check this. Here is a full truth table for these two formulas:

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>$(P \land Q) \rightarrow R$</th>
<th>$P \rightarrow R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment 1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Assignment 2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Assignment 3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td><strong>Assignment 4</strong></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>Assignment 5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Assignment 6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Assignment 7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Assignment 8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

As you can see, only in **Assignment 4** do we have the premise true and the conclusion false.
§11. Equivalence, Truth Functions and Adequacy

11.1 Equivalences

We can provide a long list of important equivalences. Some of them have names.

\[
\begin{align*}
A \land B &\equiv B \land A \quad \text{Commutativity of } \land \\
A \lor B &\equiv B \lor A \quad \text{Commutativity of } \lor \\
\neg(A \land B) &\equiv \neg A \lor \neg B \quad \text{De Morgan's Law} \\
\neg(A \lor B) &\equiv \neg A \land \neg B \quad \text{De Morgan's Law} \\
\neg \neg A &\equiv A \quad \text{Double Negation} \\
A \rightarrow B &\equiv \neg A \lor B \quad \text{Contraposition} \\
\neg(A \rightarrow B) &\equiv A \land \neg B \\
A \leftrightarrow B &\equiv (A \land B) \lor (\neg A \land \neg B) \quad \text{Commutativity of } \leftrightarrow \\
A \leftrightarrow B &\equiv (A \rightarrow B) \land (B \rightarrow A) \\
\neg(A \leftrightarrow B) &\equiv A \leftrightarrow \neg B \\
A \land (B \land C) &\equiv (A \land B) \land C \quad \text{Associativity of } \land \\
A \lor (B \lor C) &\equiv (A \lor B) \lor C \quad \text{Associativity of } \lor \\
A \land (B \lor C) &\equiv (A \land B) \lor (A \land C) \quad \text{Distributivity of } \land \text{ over } \lor \\
A \lor (B \land C) &\equiv (A \lor B) \land (A \lor C) \quad \text{Distributivity of } \lor \text{ over } \land \\
A \rightarrow (B \rightarrow C) &\equiv (A \land B) \rightarrow C \quad \text{Importation}
\end{align*}
\]

Each of these can be established using either a truth table or using a semantic tableau.

11.2 Truth Functions

We can think of a function as an abstract machine which, given an input, determines an output. (In a related fashion, we can also think of a function as a “flow diagram”). For example, we can think of the “square-of” function as the machine which, whenever given a number \(x\) as input, always gives its square \(x^2\) as output. We can indicate this idea as follows:

\[
\begin{array}{ccc}
\text{x} & \xrightarrow{\text{SQUARE-OF FUNCTION}} & \text{x}^2
\end{array}
\]

In general, for a function \(f\), we have the diagram:

\[
\begin{array}{ccc}
\text{x} & \xrightarrow{f} & f(x)
\end{array}
\]
(i.e., when the input to the function \( f \) is \( x \), the output is \( f(x) \)).

The input to a function is sometimes called the \textbf{argument}. The output of a function for a particular argument is called is \textbf{value}. A function can have more than one input. The addition function \( + \) and the multiplication function \( \times \) are both functions which have \textbf{two arguments}. Thus,

\[
\begin{array}{ccc}
\times & \downarrow & \\
x & \rightarrow & x \times y \\
y & \rightarrow & \\
\end{array}
\]

\textbf{Definition:} A \textbf{truth function} is a function whose only inputs and outputs are the truth values \( T \) and \( F \).

The logical connectives \( \neg \), \( \land \), \( \lor \), \( \rightarrow \) and \( \leftrightarrow \) represent truth functions. These truth functions are given by their truth tables.

For example, the negation truth function (which is represented by \( \neg \)) is a one-argument function which maps \( T \) to \( F \) and maps \( F \) to \( T \).

Diagrammatically, we could indicate this as follows:

\[
\begin{array}{ccc}
\neg & \downarrow & \\
P & \rightarrow & \neg P \\
\end{array}
\]

Here we think of the input \( P \) has having the possible values \( T \) and \( F \).

The \textbf{conjunction truth function} (represented by \( \land \)) is a two-argument function which maps the pair of arguments \((T, T)\) to \( T \) and maps other pairs to \( F \).

\[
\begin{array}{ccc}
\land & \downarrow & \\
P & \rightarrow & P \land Q \\
Q & \rightarrow & \\
\end{array}
\]

11.3 How Many Truth Functions are There?

An important theoretical question is this: “How many distinct truth functions are there for \( n \) arguments?” It is easy to see that for 1 argument, there are 2 assignments, and then 4 different truth functions, which we call \( J_1, J_2, J_3 \) and \( J_4 \).

Thus, here is a table of the \textbf{unary truth functions} (truth functions of one argument)

\[
\begin{array}{ccccc}
A & J_1 & J_2 & J_3 & J_4 \\
T & T & T & F & F \\
\end{array}
\]
Thus, for example, \( J_3 \) is the \textbf{negation truth function} (represented by \( \neg \)).

Similarly, for \textbf{binary truth functions} (2 arguments), there are \( 2^2 = 4 \) assignments and \( 2^4 = 16 \) truth functions. Let’s list all these binary truth functions, of 2 arguments.

\[
\begin{array}{ccccccc}
A & B & K_5 & K_6 & K_7 & K_8 & K_9 & K_{10} & K_{11} & K_{12} & K_{13} & K_{14} & K_{15} & K_{16} & K_{17} & K_{18} & K_{19} & K_{20} \\
T & T & T & F & T & F & T & F & T & F & T & F & T & F & T & F & T & F \\
T & F & T & T & F & T & F & T & F & T & F & T & F & T & F & T & F & T \\
F & T & T & T & T & T & F & F & F & T & T & T & F & T & F & F & F & F \\
F & F & T & T & T & T & T & T & T & F & F & F & F & F & F & F & F & F \\
\end{array}
\]

As you can see, the truth function \( K_{13} \) is the same as \( \lor \), the truth function \( K_{19} \) is the same as \( \land \); and \( K_7 \) is the same as \( \rightarrow \) and \( K_{11} \) is the same as \( \leftrightarrow \).

In a moment, we shall show that there is something quite special about the truth functions \( K_6 \) (NAND) and \( K_{12} \) (NOR).

For 3 arguments, there are \( 2^3 = 8 \) assignments, and \( 2^8 = 256 \) truth functions. In general, for \( n \) arguments, there are \( 2^n = k \) assignments and \( 2^k \) truth functions.

As you can see, there are lots of different truth functions (actually, infinitely many).

\textbf{11.4 Defining Truth Functions Using \( \neg \), \( \land \) and \( \lor \)}

So, what’s so special about \( \neg \), \( \land \) and \( \lor \)? Well, it turns out that:

\textbf{Any truth function can be defined using just \( \neg \), \( \land \) and \( \lor \).}

We say that:

The set \( \{ \neg, \land, \lor \} \) is an \textbf{adequate set of connectives}.

In fact, it will turn out that \( \{ \neg, \land \} \) or \( \{ \neg, \lor \} \) are both adequate. However, some sets of connectives are not adequate. For example, \( \{ \land, \lor \} \) is not adequate.

The truth functions represented by the connectives \( \rightarrow \) and \( \leftrightarrow \) can be \textbf{defined} using \( \{ \neg, \land, \lor \} \). These definitions are given by \textbf{truth-functional equivalences}.

What do we mean by \textbf{definable}?

\textbf{Definition:} A 2-place connective \( F \) is \textbf{definable} using the set of connectives \( \{ F_1, F_2, \ldots \} \) just in case the formula \( F(A, B) \) is equivalent to some formula \( \ldots A \ldots B \ldots \), where the expression \( \ldots A \ldots B \ldots \) contains only connectives from the set \( \{ F_1, F_2, \ldots \} \).

For example, by using truth tables or semantic tableaux, you can prove that,

\[
\begin{align*}
(i) & \quad A \rightarrow B & = & \neg A \lor B \\
(ii) & \quad A \leftrightarrow B & = & (A \land B) \lor (\neg A \land \neg B) 
\end{align*}
\]

Thus, both \( \rightarrow \) and \( \leftrightarrow \) are \textbf{definable} using \( \{ \neg, \land, \lor \} \).
Moreover, you can easily show that,

(iii) \( A \land B = \neg(\neg A \lor \neg B) \)

(iv) \( A \lor B = \neg(\neg A \land \neg B) \)

So \( \land \) is definable from the set \( \{\neg, \lor\} \).
And similarly \( \lor \) is definable from the set \( \{\neg, \land\} \).

11.5 Adequate Sets of Connectives

It turns out that: Every truth function can be defined using just \( \neg \), \( \land \) and \( \lor \). In order to prove this, one uses the Disjunctive Normal Form Theorem. (We shall not prove this here.) This theorem states that if \( A \) is an arbitrary formula, representing any \( n \)-place truth function you like, then you can find a “disjunctive normal form” for \( A \): this formula will be built only from \( \neg \), \( \land \) and \( \lor \) and it will be logically equivalent to \( A \).

We say that the set \( \{\neg, \land, \lor\} \) is an adequate set of connectives.

**Definition:** A set of connectives is called adequate just in case every truth function can be defined from it.

Indeed, since \( \lor \) is definable from \( \neg \) and \( \land \), it follows that \( \{\neg, \land\} \) is an adequate set.

Similarly, since \( \land \) is definable from \( \neg \) and \( \lor \), it follows that \( \{\neg, \lor\} \) is an adequate set.

[Exercise: Show how to define \( \land \) from \( \neg \) and \( \to \). Thereby, conclude that \( \{\neg, \to\} \) is an adequate set also.]

However, some sets of connectives are not adequate. For example, the set \( \{\land, \lor\} \) is not adequate. You cannot define negation \( \neg \) using \( \{\land, \lor\} \).

We show below that one can find a single 2-place connective which is adequate. That is, a single connective \( K \) such that any logical connective (representing any truth function) can be defined using just \( K \) alone. In fact, there are two of these. (Indeed, there are exactly two. No other single connective is, on its own, adequate.)

11.6 Two New Connectives: NAND (\( | \)) and NOR (\( \downarrow \))

Every truth functional connective can be defined using \( \{\neg, \land, \lor\} \).

Let us define two new truth-functional binary connectives:

**NAND (“not … and …”)**

\[
\text{NAND}(A, B) \quad \text{written } A | B \quad = \quad \neg(A \land B)
\]

**NOR (“not … or …”):**

\[
\text{NOR}(A, B) \quad \text{written } A \downarrow B \quad = \quad \neg(A \lor B)
\]

Given these definitions we can quickly figure out their truth tables:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>A ↓ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>
As you can see, these correspond to the truth functions we called $K_6$ and $K_{12}$ above. From these truth tables you can figure out **tableau rules** for $A | B$ and $A \downarrow B$. They are:

**Tableau Rules for $|$:**

| $A | B$ | $\neg (A | B)$ |
|------|------|----------------|
| /    | \   | $\neg A$       |
| \   | /   | $\neg B$       |

**Tableau Rules for $\downarrow$:**

<table>
<thead>
<tr>
<th>$A \downarrow B$</th>
<th>$\neg (A \downarrow B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>\</td>
</tr>
<tr>
<td>\</td>
<td>/</td>
</tr>
</tbody>
</table>

$\neg A$ $\neg B$

$A$

$B$

11.7 NAND ( | ) Alone is Adequate!

Now we show the fact that **every truth functional connective** can be defined using $|$ alone (i.e., using NAND). In order to do this, we simply need to show how to define **negation** $\neg$ and **conjunction** $\wedge$ from $|$. Since we already know that $\{\neg, \wedge\}$ is adequate, it immediately follows that $|$ is adequate.

The crucial trick is to define **negation** $\neg$ using $|$. This is quite easy (when you understand it).

Consider the truth table for any formula of the form $A | A$.

| $A$ | $A | A$ |
|-----|------|
| T   | F    |
| F   | T    |

That is, we see that

$$ (1) \quad \neg A \equiv A | A $$

[A similar truth table will reveal that: $\neg A \equiv A \downarrow A$.]

Next we want to define $\wedge$ from $|$. Again this is possible.

Note that it follows from the definition of $|$ that $A \wedge B \equiv \neg (A | B)$. Thus,

$$ (2) \quad A \wedge B \equiv \neg (A | B) \equiv (A | B) \downarrow (A | B) $$

I.e., we have used the definition of $\neg$ above to convert $\neg(\ldots)$ to $(\ldots) | (\ldots)$. 


The results (1) and (2) mean that both ¬ and ∧ can be defined using | alone. It follows that | alone is adequate.

Finally, how do we define ∨? That is, how do we find a formula equivalent to A ∨ B?

Well, A ∨ B is equivalent to ¬(¬A ∧ ¬B) and this is equivalent to ¬((A|A) ∧ (B|B)). This is equivalent to (A|A)|(B|B). So, we have

\[(3) \quad A \lor B \equiv \neg((A \mid A) \land (B \mid B)) \equiv (A \mid A) \mid (B \mid B)\]

Exactly analogous reasoning shows that ↓ is also adequate.

### 11.8 Tableaux Using NAND (|) and NOR (↓)

A propositional language containing the connective | (or ↓, or both) can easily be considered. E.g., L[P, Q, R, ..., |, ¬, ∧]. Again, the tableau method for | and ↓ works exactly as before. Thus, if a finished tableau generated from an initial list Δ is closed, then Δ is inconsistent (i.e., Δ has no models). And if a finished tableau generated from an initial list Δ is open, then Δ is consistent (i.e., Δ has at least one model, and this is just the branch model associated with any open finished branch in the tableau).

**Example 1**: Prove that the set Δ = {P | Q, P, Q} is inconsistent.

1. P | Q
2. P
3. Q
4. ¬P ¬Q

The tableau is closed. It follows that the initial list Δ = {P | Q, P, Q} is inconsistent.

**Example 2**: Prove that (P | Q) | (P | Q) ⊢ P ∧ Q.

1. (P | Q) | (P | Q) initial list
2. ¬(P ∧ Q) initial list
3. ¬(P | Q) ¬(P | Q) | 1
4. P P ¬| 3
5. Q Q ¬| 3
6. ¬P ¬Q ¬P ¬Q ¬∧ 2

\[\Box \quad \Box \quad \Box \quad \Box \quad \Box \]

76
§12. Soundness, Completeness and Decidability

12.1 Metalogic

In this final lecture on Propositional Logic, we discuss some basic metalogical facts about propositional logic. Most of what is called mathematical logic in fact consists in what is sometimes called “metalogic”. Logicians rarely spend time constructing formal proofs! Rather, they informally prove mathematical facts about logical systems.

The basic metalogical properties of propositional logic concern soundness, completeness and decidability. The purpose of this lecture is to give a taste of this branch of mathematical logic.

12.2 Informal Ideas of Soundness and Completeness

Consider any source of information, such as The Guardian. We might say that,

*The Guardian* is **sound** if and only if for any proposition \( p \), if *The Guardian* says \( p \), then \( p \) (is the case)

*The Guardian* is **complete** if and only if for any proposition \( p \), if \( p \) (is the case), then *The Guardian* says \( p \).

More generally, a person (book, story, method, etc.) is **sound** just in case everything they say, or believe, is true.

A person (book, story, method, etc.) is **complete** just in case, if something is the case, they do in fact say it (or believe it).

Of course, soundness and completeness are presumably impossible properties of ordinary mortals. We are often unsound (we make mistakes), and we are obviously hugely incomplete (there are lots of truths we don’t know).

12.3 Soundness and Completeness for the Tableau Method

Let \( A_1, \ldots, A_n \) be a set of formulas and let \( B \) be a formula. We have introduced the tableau method to test whether a sequent,

\[ A_1, \ldots, A_n : B \]

is valid or not.

The method consists in forming the counter-example set \( \{ A_1, \ldots, A_n, \neg B \} \) and developing a tableau by following the tableau rules.

Throughout the logic course we have simply assumed:

**SOUNDNESS of the Tableau Method:**

If a tableau for \( \{ A_1, \ldots, A_n, \neg B \} \) is closed, then the sequent is valid.

**COMPLETENESS of the Tableau Method:**

If a tableau for \( \{ A_1, \ldots, A_n, \neg B \} \) is finished and open, the sequent is invalid.

Schematically,

**SOUNDNESS:** \( \text{Closed Tableau} \Rightarrow \text{Validity} \)

**COMPLETENESS:** \( \text{Finished Open Tableau} \Rightarrow \text{Invalidity} \)
If the tableau system is sound, then this means that if a tableau is closed, then the corresponding sequent is indeed valid. That is, a closed tableau “says that” the corresponding sequent is valid.

If the tableau system is complete, then this means that if a tableau is finished and open, then the corresponding argument is indeed invalid. So, an open finished tableau “says that” the corresponding argument is invalid.

12.4 Soundness and Completeness of the Tableau System

12.4.1 Logical Consequence and Validity

We write,

\[ A_1, \ldots, A_n \models B \]

to mean

B is a logical consequence of \( A_1, \ldots, A_n \).

That is,

B is true in any assignment which makes \( A_1, \ldots, A_n \) true.

This is equivalent to saying

The sequent \( A_1, \ldots, A_n : B \) is valid.

12.4.2 Tableau Deducibility/Provability

We write:

\[ A_1, \ldots, A_n \vdash B \]

to mean:

There is a closed tableau with initial list \( \{A_1, \ldots, A_n, \neg B\} \).

We also say that:

B is provable/deducible from \( A_1, \ldots, A_n \) using the tableau method.

12.4.3 Consequence and Provability

Notice that validity and deducibility (or provability) are concepts which are defined differently.

Validity is a semantic notion (defined in terms of truth assignments).

Deducibility/provability is a syntactical notion (defined in terms of closure of the tableau).

However, the soundness and completeness theorems tell us that these concepts coincide exactly.
12.4.4 Expressing Soundness and Completeness

We can now express the Soundness and Completeness claims above:

**SOUNDNESS of the Tableau Method:**

If $A_1, \ldots, A_n \vdash B$ then $A_1, \ldots, A_n \models B$.

**COMPLETENESS of the Tableau Method:**

If $A_1, \ldots, A_n \models B$ then $A_1, \ldots, A_n \vdash B$.

12.5 Demonstrating Soundness and Completeness

We shall postpone this until Lecture 24, after we have discussed predicate logic.

12.6 Decidability of Propositional Logic

A property $P$ of formulas is called a **decidable property** just in case there is a "mechanical" procedure such that, for any given input formula $A$,

(i) if $A$ has the property $P$, then the procedure demonstrates in finitely many steps that $A$ has the property,

(ii) if the formula does not have the property $P$, then the procedure demonstrates this also.

We say that the property $P$ is **decidable**.

Let $A$ be an expression built from expressions of some propositional language $L$. Consider the property of **being a formula**. It turns out that this property is decidable. So, if $A$ is an expression, we can decide in finitely many steps whether or not $A$ is a formula or not.

Similarly, let $A$ be a formula and consider the property of being a **tautology**. To test if $A$ is a tautology, we negate $A$ and form a tableau for $\neg A$. Of course, any such tableau must be finite. For there is an upper bound on the number of sentence letters present in $A$. As we apply the tableau rules, the formula gets “broken down” into smaller and smaller parts, and eventually this must terminate. So, either the tableau is closed, or some branch is finished and open. In particular, every branch (finished or open) must be finitely long.

It follows from this that:

The property of being a tautology is decidable.

Given any propositional formula $A$, we can decide in a finite number of steps whether or not it is a tautology.

This also generalizes to arbitrary **sequents**.

Let $A_1, \ldots, A_n : B$ be a sequent of propositional formulas. It may be valid or invalid.

If we construct a tableau from the counter-example set \{$A_1, \ldots, A_n, \neg B$\} as an initial list, it is again easy to see that any such tableau must be finite. For there is an upper bound on the number of sentence letters present in the formulas $A_1, \ldots, A_n, \neg B$. As we apply the tableau rules, each formula gets “broken down” into smaller and smaller parts, and eventually this must terminate.
So, either the tableau is closed, or some branch is finished and open. In particular, every branch (finished or open) must be **finitely long**.

It follows that, given any propositional sequent $A_1, \ldots, A_n : B$, we can decide in a **finite number of steps** whether or not it is valid.

So,

**The property of being a valid propositional sequent is decidable.**

In short,

**Propositional Logic is decidable.**

As we shall show later, this important property does **not** hold for predicate logic.

### 12.7 Other Deductive Systems

The **semantic tableau method** is one of several methods used for demonstrating that sequents are valid. **However, they are all equivalent in that they yield exactly the same results:** that is, the same sequents are proved valid.

There are two main alternatives:

1. **(i) Axiomatic Proofs**
2. **(ii) Natural Deduction/Sequent Calculus.**

#### 12.7.1 Axiomatic Proofs

Until 1930 or so, following Frege, Russell and Hilbert, logical deduction was always presented by **axiomatic proofs**. This method is unfortunately very cumbersome.

The main idea is this

(a) One presents a system of **logical axioms**.

(b) One has a number of **rules of inference**.

Then, the **proof** of a sequent $A_1, \ldots, A_n : B$

consists of sequence of formulas

$F_1, F_2, \ldots, F_k$

where $F_k$ is the formula $B$, and where each $F_1, F_2, \ldots,$ is either one of the assumptions $A_1, \ldots, A_n$ or is a substitution instance of one of the logical axioms, or is obtained by using a rule of inference on some earlier $F_i$’s.

Such a sequence of formulas is called a **proof sequence**.

Of course, a proof sequence can be written vertically, as follows:

1. $F_1$
2. $F_2$
   ....
   ....

$k$ $F_k$ (i.e., $B$)
As an example of this, here is such an axiomatic proof system for logic using just the connectives $\neg$ and $\rightarrow$ (one can show that these two connectives are adequate).

**Logical Axioms (strictly speaking: Axiom Schemes)**

(I) $A \rightarrow (B \rightarrow A)$

(II) $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

(III) $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$

**Rule of Inference**

*Modus Ponens.* If you have formulas $A$ and $A \rightarrow B$ at some point in the proof, then you can add $B$ at a later point in the proof.

For example, here is an axiomatic demonstration for $\vdash P \rightarrow P$.

1. $(P \rightarrow ((P \rightarrow P) \rightarrow P)) \rightarrow ((P \rightarrow (P \rightarrow P)) \rightarrow (P \rightarrow P))$ instance of (II)
2. $P \rightarrow ((P \rightarrow P) \rightarrow P)$ instance of (I)
3. $(P \rightarrow (P \rightarrow P)) \rightarrow (P \rightarrow P)$ MP 1, 2
4. $P \rightarrow (P \rightarrow P)$ instance of (I)
5. $P \rightarrow P$ MP 3, 4

Not very enlightening?

As noted at the beginning of the section, most of ‘real’ logic is *metalogic*, and logicians don’t spend much time carrying out object level proofs, such as the above. So even though the axiomatic system is highly cumbersome to prove things in, this is made up for by the fact that it is a very streamlined and convenient system to prove things about, at the meta-level.

**12.7.2 Natural Deduction**

A rather more enlightening method of proving sequents is given by the system of *Natural Deduction* (and the related system of Sequent Calculus).

Natural deduction proofs are called *derivations*. They differ from axiomatic proofs in that they involve no special logical axioms.

Roughly, the system of natural deduction involves 12 or so rules governing how to reason with the various connectives. These rules are all sound rules (technically, they map valid sequents to valid sequents).

As an illustration, consider how we might prove

$$P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R$$

We proceed as follows
1. \( P \rightarrow Q \) \( \vdash \) \( P \rightarrow Q \) Rule of Assumptions
2. \( Q \rightarrow R \) \( \vdash \) \( Q \rightarrow R \) Rule of Assumptions
3. \( P \) \( \vdash \) \( P \) Rule of Assumptions
4. \( P \rightarrow Q, P \) \( \vdash \) \( Q \) Modus Ponendo Ponens 1, 3
5. \( P \rightarrow Q, Q \rightarrow R, P \) \( \vdash \) \( R \) Modus Ponendo Ponens 2, 4
6. \( P \rightarrow Q, Q \rightarrow R \) \( \vdash \) \( P \rightarrow R \) Conditional Proof 3, 5

We have used three different Natural Deduction rules: Rule of Assumptions, Modus Ponendo Ponens and Conditional Proof (see below).

The above derivation can be rewritten as follows:

<table>
<thead>
<tr>
<th>Premises</th>
<th>Line Number</th>
<th>Formula</th>
<th>Rule Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( P \rightarrow Q )</td>
<td>(1)</td>
<td>( P \rightarrow Q )</td>
<td>Rule of Assumptions</td>
</tr>
<tr>
<td>2. ( Q \rightarrow R )</td>
<td>(2)</td>
<td>( Q \rightarrow R )</td>
<td>Rule of Assumptions</td>
</tr>
<tr>
<td>3. ( P )</td>
<td>(3)</td>
<td>( P )</td>
<td>Rule of Assumptions</td>
</tr>
<tr>
<td>4. ( P \rightarrow Q, P )</td>
<td>(4)</td>
<td>( Q )</td>
<td>Modus Ponendo Ponens 1, 3</td>
</tr>
<tr>
<td>5. ( P \rightarrow Q, Q \rightarrow R, P )</td>
<td>(5)</td>
<td>( R )</td>
<td>Modus Ponendo Ponens 2, 4</td>
</tr>
<tr>
<td>6. ( P \rightarrow Q, Q \rightarrow R )</td>
<td>(6)</td>
<td>( P \rightarrow R )</td>
<td>Conditional Proof 3, 5</td>
</tr>
</tbody>
</table>

However, such derivations are not as easy to learn and construct as are semantic tableaux. One reason is that the system of rules is quite complicated.

Here is a full list of the system of rules for Natural Deduction:

**System of Rules for Natural Deduction**

Here, \( A \), \( B \) and \( C \) can be any formulas you like. \( \Delta \), \( \Sigma \) and \( \Phi \) can be any sets of formulas you like.

1. **Rule of Assumptions**
   \[ A \vdash A \]

2. **\( \wedge \)-Introduction (\( \wedge \)-I)**
   \[ \Delta \vdash A \]
   \[ \Sigma \vdash B \]
   \[ \Delta, \Sigma \vdash A \wedge B \]

3. **\( \wedge \)-Elimination (\( \wedge \)-E)**
   \[ \Delta \vdash A \wedge B \]
   \[ \Delta \vdash A \]
   \[ \Delta \vdash B \]

4. **Modus Ponendo Ponens (MPP)**
   \[ \Delta \vdash A \rightarrow B \]
   \[ \Sigma \vdash A \]
   \[ \Delta \cup \Sigma \vdash B \]

5. **Conditional Proof (CP)**
   \[ \Delta, A \vdash B \]
   \[ \Delta \vdash A \rightarrow B \]
6. $\lor$-Introduction ($\lor$-I)

\[ \Delta \vdash A \quad \Delta \vdash A \]
\[ \Delta \vdash A \lor B \quad \Delta \vdash B \lor A \]

7. $\lor$-Elimination

\[ \Delta \vdash A \lor B \]
\[ \Sigma, A \vdash C \]
\[ \Phi, B \vdash C \]
\[ \Delta \cup \Sigma \cup \Phi \vdash C \]

8. $\neg \neg$-Introduction (DNI)

\[ \Delta \vdash A \]
\[ \Delta \vdash \neg \neg A \]

9. $\neg \neg$-Elimination (DNE)

\[ \Delta \vdash \neg \neg A \]

10. Modus Tollendo Tollens (MTT)

\[ \Delta \vdash A \rightarrow B \]
\[ \Sigma \vdash \neg B \]
\[ \Delta \cup \Sigma \vdash \neg A \]

11. Disjunctive Syllogism (DS)

\[ \Delta \vdash A \lor B \]
\[ \Sigma \vdash \neg A \]
\[ \Sigma \vdash \neg B \]
\[ \Delta \cup \Sigma \vdash B \]
\[ \Delta \cup \Sigma \vdash A \]

12. Reductio Ad Absurdum (RAA)

\[ \Delta, A \vdash B \land \neg B \]
\[ \Delta \vdash \neg A \]

12.8 Equivalence of These Systems

All of these different systems of formalizing logical reasoning are equivalent. They validate exactly the same sequents. So, if we write

\[ A_1, \ldots, A_n \vdash_{\text{Tab}} B \]
There is a tableau proof of B from $A_1, \ldots, A_n$.

\[ A_1, \ldots, A_n \vdash_{\text{Axiom}} B \]
There is an axiomatic proof of B from $A_1, \ldots, A_n$.

\[ A_1, \ldots, A_n \vdash_{\text{ND}} B \]
There is an ND derivation of B from $A_1, \ldots, A_n$.

Then it turns out that these are all equivalent.

\[ A_1, \ldots, A_n \vdash_{\text{Tab}} B \]
iff \[ A_1, \ldots, A_n \vdash_{\text{Axiom}} B \]
iff \[ A_1, \ldots, A_n \vdash_{\text{ND}} B \]