

Proving Some Theorems of Propositional Logic in Lean

By: Aqil Azmi

Submitted as the junior-year expository mathematics paper for the mathematics concentration requirement

1. Introduction and General Mathematical Remarks

Propositional logic, also known as *zeroth-order logic*, is one of the most basic formal languages of logic. Propositional logic deals with statements that can either be true or false, and can be combined with each other through logical connectives to create more complex statements. The formal system we are working on in this paper is known as the *Hilbert system*, which relies on a few axioms known as the *Hilbert axioms* (see Axiom 6), and a single rule of inference known as *modus ponens* (see Definition 5). Due to its minimalist nature, it is a rigorous and axiomatic language that can be used to formalise much of mathematics. In this sense, it is also a suitable candidate to be implemented in Lean which requires the same amount of rigour.

In this paper, our main focus is on two of the most important theorems for any logical system, known as *soundness* (see Theorem 2) and *completeness* (see Theorem 3). In basic terms, soundness guarantees that a logical system will never lead to a false conclusion given a set of true premises while completeness guarantees that all valid conclusions that can be derived from a set of premises can actually be derived. These two properties are important because it shows that the logical system is reliable and trustworthy. In our case, the Hilbert system being reliable allows higher order logic to be built on it by extending it by introducing quantifiers, which can give us first-order logic known as *predicate logic* and second-order logic, and even higher orders.

In particular, understanding the completeness theorem for propositional logic is essential because much of the same reasoning is used to prove completeness for higher order logic. This will then allow us to see that certain logical systems, such as second-order logic, fails completeness, and when we generalise this to other logical systems that are more advanced, we are able to prove and understand one of the most important theorems in mathematical logic, and arguably one of the most important theorems of the 20th century, which is Gödel's Incompleteness Theorem. In basic terms, it posits that a logical system that is powerful enough to represent certain basic mathematical properties will necessarily be incomplete. This means that there are true statements that cannot be proven by that system.

In Section 2, we give a brief introduction of what the Lean Theorem Prover is. In Section 3, we will look at key ideas in implementing these theorems in Lean. In Section 4, we outline the mathematical proof for these theorems which we rely on when proving them in Lean. Finally, in Section 5, we discuss some future improvements that one can take in implementing these theorems.

2. Lean Theorem Prover

Lean 3.0 is a programming language designed as a theorem prover for mathematics. In using it, most people rely on its online library that the community continues to expand called *mathlib* which includes a lot of basic mathematical tools, functions and theorems once it is imported. The language fundamentally relies on what is called *types* because it is a language known as

dependent type theory. What this means is that each mathematical object that we use in our code must be given a specific type, and in turn, every function that we use or define only takes a certain type as input and outputs a certain type. This constant reliance on types means that one cannot make any unjustified *jumps* in reasoning like how one would on paper, and in turn, it means that every step of a proof in Lean must be fully justified, which gives it an increase in rigour.

3. Comments on Lean Implementation

1. Types

A key step that simplifies a lot of implementing propositional logic in Lean is that, unlike a course in logic, we are not required to go through truth-tables for each logical connectors because Lean already has all of them built in in *mathlib*. However, we have to decide what *type* to make our atomic sentences. In my implementation of the Deduction Theorem and Soundness Theorem, I used *Prop* as the type for atomic sentences, and combining them with others through the built-in logical connectives also yields a type *Prop*. Thus, working with this is much simpler because Lean makes no distinction between an atomic sentence and a formula, and one can use them both interchangeably. However, this becomes a problem in my implementation of the Completeness Theorem since it requires us to be able to access each atomic sentence in any formula. Therefore, in this part of my implementation, I had to create a new type called *formula* which is inductively defined, and is built from a smaller type I defined as *atom*. The key for *atom* is to define it inductively from natural numbers, so that we can map each unique atomic sentence to a particular natural number.

2. Truth tables

Another key aspect of this implementation is that we need to be able to evaluate the truth-value of a complex formula, and be able to compare the truth-values for different valuations in order to say whether they are equivalent, valid, satisfiable, and tautological. In Definition 7, we call this *valuation*, and in our implementation, it is simply a function that changes the type of objects to become type *bool*. *bool* is similar to type *Prop* where it can have only true or false values, but unlike *Prop*, it is not concerned with the composition of the formula, but only with its truth-assignment. This is then used in our definition of semantic implication, which compares the truth values of proposition to say that one implies the other. This is also used in Completeness Theorem where we changed *atom* to *bool*.

3. Entailment

As we shall see in Section 4, all three theorems rely on the notion of syntactic entailment. Since entailment simply means that one is able to derive a conclusion, I defined a function called *derivation* that simply takes a set of *Prop* (being our set of premises) and outputs a type *Prop* if a certain proposition can be derived from the set. By defining this inductively, I am able to specify specifically what rules of inferences I include in *derivation*. This is incredibly useful because not only am I able to specify I want to work with the Hilbert system, but in the case where I want to add more rules of inferences to the system, I simply have to add them to the definition of *derivation*.

4. Length of formulae

In our proof of the Completeness Theorem (see Theorem 3), we rely a lot on the length of a formula, i.e., how many atomic sentences and logical connectives it has. A key step in our proof

in Lemma 10 is applying induction on the number of logical connectives in a formula, and the key step in the proof of the theorem is eliminating a number of propositions step by step from the set of premises. To implement this, I had to define a new object called *degree* for this which simply changes the *formula* back to a natural number.

5. Lists

This is arguably the most important idea of implementation for Completeness. I started by using *strings* and was halfway done before I realised it did not suffice my needs. Therefore, I found it incredibly useful to encode the atomic sentences as a list within the formula. To do this, I defined the object *atomic_sentences* to change the type from a *formula* to *list atom*.

4. Proof

Theorem 1 (Deduction Theorem) *Let A and B denote a formula. B is deducible from the set of assumptions $\Gamma \cup \{A\}$ if and only if the implication $A \rightarrow B$ is deducible from Γ , where Γ denotes a set of propositions A_1, A_2, \dots, A_n . In other words:*

$$\Gamma \cup \{A\} \vdash B \iff \Gamma \vdash A \rightarrow B$$

Proof. First, we need to define the basic structure of propositional logic.

Definition 1 (Proposition) A proposition is a statement that is either true or false, but not both. These are denoted by propositional variables such as “p”, “q” and “r”. We shall call these propositional variables *atomic sentences* to indicate that they are the most basic unit.

Definition 2 (Formula) A formula is a composition of atomic sentences and logical connectors. They also contain a truth-value which are evaluated according to the truth-values of each atomic sentence contained within them and according to the relationship of these truth-values over each logical connector.

Definition 3 (Logical connector) A logical connector allows multiple atomic sentences to be compounded to form formulae. In propositional logic, we will use “and”, “not”, “or”, “if-then” “if and only if”. We assume that the reader is familiar with the truth-tables for each connector.

Definition 4 (Deducibility) Let C be a proposition, and let S be a set of propositions. We say that C is *deducible* or *derivable* from S , denoted by $S \vdash C$, if and only if there exists a finite number or sequence of propositions, where each proposition is either an element of S , or is obtained from earlier propositions in the sequence by applying valid rules of inference, or a logical equivalence, such that the final proposition is C .

We also call this *syntactic entailment*, and we say that S (syntactically) *entails* C .

Definition 5 (Rule of inference) A rule of inference (or rule of deduction) is a method of deriving a conclusion from one or more premises, where the conclusion follows logically from the premise(s). In the most minimalist form of propositional logic, we only require two:

1. (Assumption)

$$\frac{\text{Assumption: } P}{\text{Conclusion: } Q}$$

2. (Modus ponens)

$$\frac{\text{Assumption: } P \quad \text{Assumption: } P \rightarrow Q}{\text{Conclusion: } Q}$$

Axiom 6 (Hilbert's Axioms) Hilbert's Axioms are three *tautologies*, which is a formula that is always true for all possible truth-value assignments to its atomic sentences. These axioms together with modus ponens is sufficient to derive all other theorems in propositional logic. The collection of these four things is known as a *Hilbert proof system*.

Axiom 1: $\phi \rightarrow (\psi \rightarrow \phi)$

Axiom 2: $(\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\phi \rightarrow \psi) \rightarrow (\phi \rightarrow \chi))$

Axiom 3: $(\neg\phi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \phi)$

Now we are finally ready to prove the deduction theorem.

Since we only defined two rules of inference explicitly in our system, we can consider each case individually in applying it on $\Gamma \cup \{A\} \vdash B$.

With the assumption rule, this means that we can assume the antecedent as our hypothesis, $\Gamma \cup \{A\} \vdash B$. We can then further split it into two possibilities: $\Gamma \vdash B$ and $\{A\} \vdash B$. With the modus ponens rule, since it uses only a conditional, we then have $\Gamma \cup \{A\} \vdash p \rightarrow q$ and $\Gamma \cup \{A\} \vdash p$ as our hypotheses. Therefore, in total, we have to consider three cases:

Case 1: $\Gamma \vdash B$

Applying Axiom 1, we would have $\Gamma \vdash B \rightarrow (A \rightarrow B)$. Applying modus ponens on both statements, we would derive $\Gamma \vdash A \rightarrow B$.

Case 2: $\{A\} \vdash B$

This means we have $B \in \{A\}$. This implies that $B = A$ since $\{A\}$ is a singleton set. Thus, $\Gamma \vdash A \rightarrow B$ can be rewritten as $\Gamma \vdash A \rightarrow A$. Since every statement implies itself, we know that this is tautologically true.

Case 3: $\Gamma \cup \{A\} \vdash p \rightarrow q$ and $\Gamma \cup \{A\} \vdash p$.

To figure out what we have to prove, we have to apply modus ponens to these two hypotheses which would yield $\Gamma \cup \{A\} \vdash q$. Then, applying the theorem we want to prove on this, our goal becomes $\Gamma \vdash A \rightarrow q$.

By applying the theorem to our two hypotheses, we would then yield two further hypotheses: $\Gamma \vdash A \rightarrow p \rightarrow q$ and $\Gamma \vdash A \rightarrow p$. Applying Axiom 2 onto the first of the two, we would then get $\Gamma \vdash (A \rightarrow p) \rightarrow (A \rightarrow q)$. Then, we can apply modus ponens to this hypotheses and to the second of the two previously to obtain our goal, $\Gamma \vdash A \rightarrow q$. ■

Theorem 2 (Soundness Theorem) Let Γ denote a set of propositions A_1, A_2, \dots, A_n . If Γ syntactically implies C (i.e. C is deducible from Γ), then Γ semantically implies C . In other words:
 $\Gamma \vdash C$ implies $\Gamma \models C$

Proof. First, we need to define the notion of valuation more formally:

Definition 7 (Valuation) A valuation is an assignment of truth-values to an atomic sentence or a formula, denoted by $v(A) = \text{true}$ if it assigns true to a formula A , and $v(A) = \text{false}$ if it assigns false to a formula A .

Definition 8 (Semantic implication) Let A and B be two propositions. We say that A *semantically implies* B , denoted by $A \models B$, if and only if every truth assignment that satisfies A also satisfies B , i.e. all truth assignments that make A true also make B true. This is also known as the *tautological*

entailment because it is equivalent to saying that the formula $A \rightarrow B$ is a tautology. In other words:

$$A \models B \iff \forall v, (v(A) = \text{true} \rightarrow v(B) = \text{true})$$

Since we only defined two rules of inference explicitly in our system, we can consider each case individually in applying it on $\Gamma \vdash C$.

With the assumption rule, this means that we can assume the antecedent as our hypothesis, $\Gamma \vdash C$.

With the modus ponens rule, since it uses only a conditional, we then have $\Gamma \vdash p \rightarrow q$ and $\Gamma \vdash p$ as our hypotheses. Therefore, we have to consider two cases:

Case 1: $\Gamma \vdash C$

This simply means that the proof of C relies on an assumption in Γ . This implies that there exists some truth assignment that satisfies every member of Γ and the assumption, which implies that this truth assignment also satisfies C . In other words, $\exists v, v(\Gamma) = \text{true} \rightarrow v(C) = \text{true}$. This would hold true for any valuation of v , thus we can generalise to have $\forall v, v(\Gamma) = \text{true} \rightarrow v(C) = \text{true}$. By the definition of semantic implication, this means we have $\Gamma \models C$.

Case 2: $\Gamma \vdash p \rightarrow q$ and $\Gamma \vdash p$

With the same reasoning as above, this implies that there exists some truth assignment that satisfies every member of Γ and the assumption which also satisfies $p \rightarrow q$. Applying the same reasoning, to the second hypothesis, we have some truth assignment that satisfies every member of Γ and the assumption which also satisfies p . In other words, we have $\exists v, v(\Gamma) = \text{true} \rightarrow v(p \rightarrow q) = \text{true}$. Again, with the same reasoning we would have $\exists v, v(\Gamma) = \text{true} \rightarrow v(p) = \text{true}$. Applying modus ponens to both $v(p \rightarrow q) = \text{true}$ and $v(p) = \text{true}$, we would then have $v(q) = \text{true}$ for some v . Thus, we can generalise this for all v and have $\forall v, v(\Gamma) = \text{true} \rightarrow v(q) = \text{true}$. Therefore, we would have $\Gamma \models q$. ■

Theorem 3 (Completeness Theorem) *Let Γ denote a set of propositions B_1, B_2, \dots, B_n . If Γ semantically implies A , then Γ syntactically implies A (i.e. A is deducible from Γ). In other words:*

$$\Gamma \models A \text{ implies } \Gamma \vdash A$$

Proof. This proof uses a proof construction method and relies heavily on Theorem 1 (Deduction Theorem). This proof is due to Kalmar, 1935.

Definition 9 Let A be a formula and b_1, b_2, \dots, b_n all be atomic sentences that occur in A . We define, for A, b_1, b_2, \dots, b_n and v , the corresponding formulae A', B_1, B_2, \dots, B_n as follows:

$$A' = \begin{cases} A & \text{if } v * (A) = \text{true} \\ \neg A & \text{if } v * (A) = \text{false} \end{cases}$$

and also

$$B_i = \begin{cases} b_i & \text{if } v(b_i) = \text{true} \\ \neg b_i & \text{if } v(b_i) = \text{false} \end{cases}$$

for $i = 1, 2, \dots, n$.

Lemma 10 (Main Lemma) For any formula A and an assignment variable v , if A', B_1, B_2, \dots, B_n are corresponding formulae defined by Definition 9, then:

$$B_1, B_2, \dots, B_n \vdash A'$$

Proof. (By induction). We apply induction on the degree of A , i.e., the number n of logical connectives in A .

(Base case). $n = 0$.

In the case that $n = 0$, A is an atomic sentence (i.e. it has no logical connectives) and thus consists of only a single propositional variable, say a . We then have two cases to consider: either $v * (A) = \text{true}$ or $v * (A) = \text{false}$.

Case 1: $v * (A) = \text{true}$

Clearly, based on Definition 9, this implies that $A' = A = a$ and $B_1 = a$. Substituting these into the lemma, we have to prove that $a \vdash a$.

Since we know that every proposition implies itself, then we have $\vdash a \rightarrow a$ since this is tautological. By applying Deduction Theorem to this, we then have $a \vdash a$.

Case 2: $v * (A) = \text{false}$

This implies that $A' = \neg A = \neg a$ and $B_1 = \neg a$. Substituting these into the lemma, we have to prove that $\neg a \vdash \neg a$.

By the same reasoning as above, we have $\vdash \neg a \rightarrow \neg a$. By applying Deduction Theorem to this, we then have $\neg a \vdash \neg a$.

(Inductive step). Assume that the lemma holds for any A with degree $j < n$. We now want to prove that it holds for A with degree n . There are several subcases to deal with:

Case 1: A is $\neg A_1$

In this case, A_1 has less than n connectives. By the inductive assumption, we have the formulae $A'_1, B_1, B_2, \dots, B_n$ corresponding to A_1 and the propositional variables b_1, b_2, \dots, b_n in A_1 , as defined by Definition 9, such that $B_1, B_2, \dots, B_n \vdash A'_1$. We can observe that the formulae A and $\neg A_1$ have the same propositional variables, so the corresponding formulae B_1, B_2, \dots, B_n for each are also the same for both of them. With our inductive assumption above, $B_1, B_2, \dots, B_n \vdash A'_1$, we are now going to prove that $B_1, B_2, \dots, B_n \vdash A'$. There are two cases to consider:

Case 1a): $v * (A) = \text{true}$

If this is the case, then we know by Definition 9 that $A'_1 = A_1$. By the inductive assumption $B_1, B_2, \dots, B_n \vdash A'_1$, we now obtain $B_1, B_2, \dots, B_n \vdash A_1$. In this case, $v * (A) = v * (\neg A_1) = \neg v * (\text{true}) = \text{false}$, and so $A' = \neg A = \neg \neg A_1$. Since we know that it is tautological that $\vdash (A_1 \rightarrow \neg \neg A_1)$, we have by monotonicity that $B_1, B_2, \dots, B_n \vdash (A_1 \rightarrow \neg \neg A_1)$. By $B_1, B_2, \dots, B_n \vdash A_1$ that we obtained previously and modus ponens, we have $B_1, B_2, \dots, B_n \vdash \neg \neg A_1$. In other words, this is equivalent to $B_1, B_2, \dots, B_n \vdash \neg A$, which is also equivalent to $B_1, B_2, \dots, B_n \vdash A'$.

Case 1b): $v * (A) = \text{false}$

If this is the case, then we know by Definition 9 that $A'_1 = \neg A_1$. Thus, $v * (A) = \text{true}$ so $A' = A$. Thus, from the inductive assumption above, we then have $B_1, B_2, \dots, B_n \vdash \neg A_1$, which means then we have $B_1, B_2, \dots, B_n \vdash A'$.

Case 2: A is $(A_1 \rightarrow A_2)$

In this case, $A = A(b_1, \dots, b_n)$, so there are some subsequences c_1, \dots, c_k and d_1, \dots, d_m (where $k, m \leq n$) of the sequence b_1, \dots, b_n such that $A_1 = A(c_1, \dots, c_k)$ and $A_2 = A(d_1, \dots, d_m)$. We know that A_1 and A_2 have less than n connectives, so by the inductive formula, we will obtain the appropriate formulae C_1, \dots, C_k and D_1, \dots, D_m such that $C_1, C_2, \dots, C_k \vdash A'_1$ and $D_1, D_2, \dots, D_m \vdash A'_2$, where C_1, C_2, \dots, C_k and D_1, D_2, \dots, D_m are subsequences of the formulae B_1, B_2, \dots, B_n corresponding to the propositional variables in A . By monotonicity, we then have

$B_1, B_2, \dots, B_n \vdash A'_1$ and $B_1, B_2, \dots, B_n \vdash A'_2$. There are three cases to consider:

Case 2a): $v * (A_1) = v * (A_2) = \text{true}$

If $v * (A_1) = \text{true}$ then A'_1 is A_1 , and if $v * (A_2) = \text{true}$ then A'_2 is A_2 . We also have $v * (A_1 \rightarrow A_2) = \text{true}$ and so A' is $(A_1 \rightarrow A_2)$. By the above and $B_1, B_2, \dots, B_n \vdash A'_2$, we then have $B_1, B_2, \dots, B_n \vdash A_2$. Then, by Axiom 1, we also have $\vdash (A_2 \rightarrow (A_1 \rightarrow A_2))$. Then, by monotonicity and applying modus ponens, we derive $B_1, B_2, \dots, B_n \vdash (A_1 \rightarrow A_2)$, which means $B_1, B_2, \dots, B_n \vdash A'$

Case 2b): $v * (A_1) = \text{true}, v * (A_2) = \text{false}$,

If $v * (A_1) = \text{true}$ then A'_1 is A_1 , and if $v * (A_2) = \text{false}$ then A'_2 is $\neg A_2$. In this case, we also have $v * (A_1 \rightarrow A_2) = \text{false}$ and so A' is $\neg(A_1 \rightarrow A_2)$. By the above and Definition 9, we have $B_1, B_2, \dots, B_n \vdash A_1$ and $B_1, B_2, \dots, B_n \vdash \neg A_2$. Since we know that $\vdash (A_1 \rightarrow (\neg A_2 \rightarrow \neg(A_1 \rightarrow A_2)))$ because it is tautological, we know by monotonicity and applying modus ponens twice that $B_1, B_2, \dots, B_n \vdash \neg(A_1 \rightarrow A_2)$, which means $B_1, B_2, \dots, B_n \vdash A'$.

Case 2c): $v * (A_1) = \text{false}$

If this is the case, then A'_1 is $\neg A_1$, and, regardless of whatever truth value v assigns to A_2 , we have $v * (A_1 \rightarrow A_2) = \text{true}$, and so A' is $(A_1 \rightarrow A_2)$. Therefore, by Definition 9 and above, we have $B_1, B_2, \dots, B_n \vdash \neg A_1$. We also know tautologically that $\vdash (\neg A_1 \rightarrow (A_1 \rightarrow A_2))$. By monotonicity and applying modus ponens, we have $B_1, B_2, \dots, B_n \vdash (A_1 \rightarrow A_2)$, which means $B_1, B_2, \dots, B_n \vdash A'$.

This means that we have covered all possible cases. Therefore, by induction on n , the proof of the lemma is complete. ■

To continue the proof of the theorem, first we assume that $\Gamma \models A$. Let b_1, b_2, \dots, b_n be all propositional variables that occur in A , i.e., $A = A(b_1, b_2, \dots, b_n)$.

Let $v : \text{propositional variables} \rightarrow \{\text{truth}, \text{false}\}$ be any variable of truth assignment, and let $v_A : \{b_1, b_2, \dots, b_n\} \rightarrow \{\text{truth}, \text{false}\}$ be its restriction to the formula A , i.e., $v_A = v|_{\{b_1, b_2, \dots, b_n\}}$.

Let $V_A = \{v_A : \{b_1, b_2, \dots, b_n\} \rightarrow \{\text{truth}, \text{false}\}\}$.

By Lemma 10 and the assumption that $\Gamma \models A$, any $v \in V_A$ defines formulae B_1, B_2, \dots, B_n such that $B_1, B_2, \dots, B_n \vdash A$.

The proof is based on a method of using all of $v \in V_A$ to define a process of elimination of all hypothesis B_1, B_2, \dots, B_n in the above equation, $B_1, B_2, \dots, B_n \vdash A$, to finally construct the proof of A in Γ , i.e., to prove that $\Gamma \vdash A$.

Step 1: Elimination of B_n

We observe by Definition 9 that each B_i is b_i or $\neg b_i$ depending on the choice of $v \in V_A$. In particular, $B_n = b_n$ or $B_n = \neg b_n$. We then choose two truth assignments $v_1 \neq v_2 \in V_A$ such that $v_1|_{\{b_1, \dots, b_{n-1}\}} = v_2|_{\{b_1, \dots, b_{n-1}\}}$ and $v_1(b_n) = \text{true}$ and $v_2(b_n) = \text{false}$.

Case 1: $v_1(b_n) = \text{true}$

Thus, by Definition 9 we have $B_n = b_n$. By $v_1 \mid \{b_1, \dots, b_{n-1}\} = v_2 \mid \{b_1, \dots, b_{n-1}\}$, assumption that $\Gamma \models A$ and Lemma 10 applied to v_1 , we have $B_1, B_2, \dots, B_{n-1}, b_n \vdash A$. By Deduction Theorem, we then have $B_1, B_2, \dots, B_{n-1} \vdash (b_n \rightarrow A)$.

Case 1: $v_1(b_n) = false$

Thus, by Definition 9 we have $B_n = \neg b_n$. By $v_1 \mid \{b_1, \dots, b_{n-1}\} = v_2 \mid \{b_1, \dots, b_{n-1}\}$, assumption that $\Gamma \models A$ and Lemma 10 applied to v_2 , we have $B_1, B_2, \dots, B_{n-1}, \neg b_n \vdash A$. By Deduction Theorem, we then have $B_1, B_2, \dots, B_{n-1} \vdash (\neg b_n \rightarrow A)$. We know tautologically that $(\phi \rightarrow \psi) \rightarrow ((\neg \phi \rightarrow \psi) \rightarrow \psi)$, thus we have that $\vdash (b_n \rightarrow A) \rightarrow ((\neg b_n \rightarrow A) \rightarrow A)$. By monotonicity, we have that $B_1, B_2, \dots, B_{n-1} \vdash (b_n \rightarrow A) \rightarrow ((\neg b_n \rightarrow A) \rightarrow A)$. Then, we apply modus ponens twice to $B_1, B_2, \dots, B_{n-1} \vdash (b_n \rightarrow A) \rightarrow ((\neg b_n \rightarrow A) \rightarrow A)$ and $B_1, B_2, \dots, B_{n-1} \vdash (\neg b_n \rightarrow A)$, we then derive $B_1, B_2, \dots, B_{n-1} \vdash A$.

Thus, we have eliminated B_n .

Step 2: Elimination of B_{n-1} from $B_1, B_2, \dots, B_{n-1} \vdash A$

We repeat Step 1. As with before, we have two cases to consider: $B_{n-1} = b_{n-1}$ or $B_{n-1} = \neg b_{n-1}$.

We then choose two truth assignments $w_1 \neq w_2 \in V_A$ such that

$w_1 \mid \{b_1, \dots, b_{n-2}\} = w_2 \mid \{b_1, \dots, b_{n-2}\} = v_1 \mid \{b_1, \dots, b_{n-2}\} = v_2 \mid \{b_1, \dots, b_{n-2}\}$

and $w_1(b_{n-1}) = true$ and $w_2(b_{n-1}) = false$.

As with before, we apply Lemma 10, Deduction Theorem, monotonicity, the tautology $(\phi \rightarrow \psi) \rightarrow ((\neg \phi \rightarrow \psi) \rightarrow \psi)$ and modus ponens twice to eliminate B_{n-1} just as we have eliminated B_n .

After n steps, we finally obtain that $\vdash A$, i.e., $\Gamma \vdash A$. Therefore, this ends the proof of Completeness Theorem. ■

5. Future Improvements

This is by no means a perfect or efficient implementation of these theorems in Lean. In particular, the most complicated part of this implementation is the Completeness Theorem, and this is due to the nature of the proof I have chosen to implement. While there are many methods of proving completeness for propositional logic, the reason why I chose this proof by construction is because it relies on the Deduction Theorem. The Deduction Theorem itself is an important proof on its own for a logical system, and by incorporating it in my proof of completeness, I would be able to show the significance of the Deduction Theorem in action. However, there could possibly be a completeness proof where one is not required to consider as many subcases.

Another improvement one can make is to try and include all the rest of the rules of inferences to the system, and prove that soundness and completeness holds for all these rules by induction. What I have failed to attempt is to do so by induction—I have found that when I included all of these rules to the system and tried to prove soundness, I would have needed to consider each case for each rule of inference added, which amounted to 12 in total. A standard mathematical proof would be to prove by induction on the *length* or number of lines of the derivation and thus generalise over all cases of rules. Therefore, implementing this aspect in Lean would be a substantial improvement.

Finally, having a unified definition and type for atomic sentences and formulae throughout the proof will make our implementation much tidier. Due to how I needed to redefine a new type called *atom* and *formula* in the completeness proof in Lean, I could have instead defined it at the start this way and used it for the proof of the other two theorems as well. The same also holds for the concept of entailment—I had to redefine it again when working in the Completeness Theorem part because I was working with *lists* and *formula* rather than *Prop*. However, this would have made the implementation for the first two theorems unnecessarily complicated.